Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

António Pedro
Moutinho Amaral

Representação de dados espácio-temporais com recurso a técnicas de Triangulação Compatível e Morphing

Representation of spatio-temporal data using Compatible Triangulation and Morphing techniques

Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2015

António Pedro Moutinho Amaral

Representação de dados espácio-temporais com recurso a técnicas de Triangulação Compatível e Morphing

Representation of spatio-temporal data using Compatible Triangulation and Morphing techniques

"*Believe you can and you're halfway there*"

— Theodore Roosevelt

Universidade de Aveiro
2015

Departamento de Eletrónica,
Telecomunicações e Informática

António Pedro Moutinho Amaral

Representação de dados espácio-temporais com recurso a técnicas de Triangulação Compatível e Morphing

Representation of spatio-temporal data using Compatible Triangulation and Morphing techniques

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor José Moreira, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Dias, Professor auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

O Júri / The Jury

Presidente / President	Professor Doutor André Ventura da Cruz Marnoto Zúquete Professor Auxiliar da Universidade de Aveiro
Vogais / Examiners Committee	Professora Doutora Maribel Yasmina Campos Alves Santos Professora Associada da Universidade do Minho (Arguente Principal)
	Professor Doutor José Manuel Matos Moreira Professor Auxiliar da Universidade de Aveiro (Orientador)

Agradecimentos /
Acknowledgements

É com uma enorme satisfação que faço uso deste espaço para agradecer a todos que, direta ou indiretamente, contribuíram para que fosse possível chegar a este momento.

Antes de mais, gostaria de deixar um enorme agradecimento aos meus orientadores, Professor José Moreira e Professor Paulo Dias, pela total disponibilidade durante toda a Dissertação, garantindo que nunca me faltava apoio sobretudo nas fases mais complicadas.

À minha namorada, Filipa, um agradecimento muito especial, por toda a dedicação, foste o meu grande apoio, que esteve sempre do meu lado ao longo de todo o trabalho, garantindo que eu continuava focado e, sobretudo, confiante quando eu não estava nos meus melhores dias.

Aos meus amigos de longos anos, Filipe Lopes, Paulo Moita, Pedro Cabral, Sara Marques e Mafalda Osório, que são os meus companheiros de guerra e estão sempre presentes, mesmo quando a distância não ajuda, um obrigado do tamanho da nossa amizade.

Foram muitas as amizades que fiz ao longo destes anos de faculdade, sendo impossível agradecer a todos neste espaço, no entanto também seria injusto, pelo apreço e importância que sinto, se não fizessem parte deste espaço. Assim sendo deixo um enorme obrigado ao Fábio Martins, Fábio Almeida, Rute Ferreira, Cristina Gaidão, Pedro Nunes, Diogo Silva, José António, Renato Costa, Andrei Vint, Pedro Santos, Luís Marques, André Malta, Miguel Soares, José Semedo e João Gama.

Um enorme agradecimento à Catarina Lino, pela ajuda que me deu na criação do logótipo para a interface gráfica desta Dissertação.

E como os últimos são sempre os primeiros, o maior agradecimento de todos vai para a minha família, em especial para os meus pais, António e Berta, e para o meu irmão Hugo, que são os meus ídolos e a quem um dia espero conseguir seguir como exemplo. Foram eles que me permitiram ter todas as oportunidades que tive e de chegar até onde cheguei, apoiando incondicionalmente, oferecendo todo o amor, carinho e, sobretudo, contaminando com toda a alegria existente nesta família. Um enorme obrigado!

Palavras Chave

Objetos Móveis, Representações Contínuas, Triangulação Compatível, Morphing, Interpolação Rígida

Resumo

Atualmente vivemos num mundo em constante evolução tecnológica, onde, a cada dia que passa, são desenvolvidas novas ferramentas com o objetivo de facilitar as tarefas para os seus utilizadores. No entanto, a tecnologia não existe apenas para facilitar tarefas, mas também para ajudar a monitorar fenómenos físicos. Um exemplo disso é o estudo do movimento de objetos, que são entidades onde a sua posição e forma varia ao longo do tempo. Estes objetos são utilizados para representar fenómenos geográficos, como o movimento de *Icebergs* e podem ser adquiridos através de imagens aéreas obtidas, por exemplo, por satélites ou veículos aéreos não tripulados. Estas imagens representam o estado do objeto móvel em diferentes instantes de tempo, logo são conjuntos de dados discretos onde existe informação implícita, porque não se sabe como decorreu a evolução entre dois estados. Nesta situação, podem ser utilizadas técnicas de *Morphing* para obter uma representação contínua da evolução do objeto móvel.

Nesta Dissertação são apresentadas um conjunto de ferramentas para a representação de dados contínuos a partir de dados discretos. Os dados discretos são representados sob a forma de um polígono origem e um polígono destino, que representam o objeto móvel em diferentes instantes de tempo. Com recurso a técnicas de Triangulação Compatível e *Morphing*, é gerada a sua transformação contínua, evitando intersecções e deformações, de forma a tornar a representação o mais natural possível. Os resultados dos testes realizados mostram que a utilização de técnicas de Interpolação Rígida baseadas em Triangulação Compatível podem ser uma solução interessante para representar o movimento de objetos ao longo do tempo e do espaço em, por exemplo, Sistemas de Informação Geográfica ou Bases de Dados Espácio-temporais.

Keywords

Moving Objects, Continuous Representations, Compatible Triangulations, Morphing, Rigid Interpolation

Abstract

Currently we live in a world in continuous technological evolution, where new tools are developed with the goal of making any task easier for its users. However, technology exists not only to facilitate tasks, but also to help monitoring real-world phenomena. An example of this is the study of the movement of objects, which are entities where the position and shape may change over time. These objects can represent geographic phenomena, like the movement of icebergs and can be acquired through aerial images obtained, for example, by satellites or unmanned aerial vehicles. These images represent the moving object state at different time instants. These are discrete datasets holding a large amount of implicit information, because the evolution between two states is unknown. In this situation, Morphing techniques can be used to obtain a continuous representation of the evolution of the moving object.

This Dissertation presents a set of tools for the representation of continuous data from discrete data. The discrete data are represented in the form of a source polygon and a target polygon, that represents a moving object at different time instants. Using Compatible Triangulations and Morphing techniques, a continuous representation is generated, avoiding deformations and intersections, in order to make the representation as natural as possible. The experimental results show that the combination of Rigid Interpolation with Compatible Triangulation is a promising solution to represent the movement of objects over time and space in, for example, geographical information systems or spatiotemporal databases.

Conteúdo

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

INTRODUÇÃO

1.1 MOTIVAÇÃO

Com a evolução tecnológica presente nos dias de hoje é possível recolher grandes volumes de informação geográfica ao longo do tempo. Esta informação pode ser obtida, por exemplo, através de sistemas GPS (*Global Positioning System*) que podem ser utilizados para recolher dados sobre a posição de automóveis, pessoas ou animais ao longo do tempo, ou através de imagens aéreas, obtidas por satélite ou VANT's (Veículos Aéreos Não Tripulados), que permitem monitorar fenómenos geográficos como o movimento e a fusão de *icebergs* ou a propagação de fogos florestais. Em termos de representação, é mais simples de fazer para o primeiro caso, pois a informação obtida pelo GPS pode ser modelada como pontos móveis, que representam a localização dos objetos ao longo do tempo. No segundo caso é mais complexo, uma vez que esta representação envolve modelar transformações, como a rotação e a deformação dos objetos que representam a evolução dos fenómenos geográficos ao longo do tempo. Estes fenómenos, cuja posição ou forma varia ao longo do tempo, são denominados de objetos móveis [1].

Contudo, a integração de objetos móveis em Sistemas de Informação Geográfica (SIG) ou Bases de Dados Espácio-temporais ainda é um desafio. *Güting et al* [1] propôs uma abordagem onde os objetos móveis são representados através de Tipos Abstractos de Dados (TAD). Esta representação, além de permitir um nível de abstração mais alto, pode ser integrada em Sistemas de Gestão de Bases de Dados, como o *PostgreSQL* e *Oracle* que possuem as extensões espaciais *PostGIS* e *Spatial*, respetivamente. Este tipo de abordagem serviu como base nas propostas de *Pelekis et al* [2], *Zhao et al* [3] e *Matos et al* [4], no entanto estas focam-se apenas na modelação e interrogação de dados, não sendo considerados aspetos

como a qualidade das representações dos objetos móveis.

Na representação de objetos móveis existem dois passos principais: o primeiro consiste em fazer uma aquisição das representações geométricas dos objetos móveis, por exemplo, a partir de uma sequência de imagens aéreas, e estabelecer uma correspondência entre os vértices das representações geométricas obtidas em pares de imagens consecutivas, problema que é conhecido como *Vertex Correspondence Problem* [5]. O segundo passo consiste em estimar uma representação contínua do comportamento espácio-temporal do objeto móvel, uma vez que as imagens aéreas representam a sua evolução em instantes pré-definidos. Para isso é necessário representar a transformação dos objetos móveis entre os momentos discretos, com recurso a técnicas de *Morphing*, sendo este problema conhecido como *Vertex Path Problem* [6].

A principal dificuldade neste processo está em obter representações realistas do objeto móvel ao longo do tempo e que, simultaneamente, formem topologias válidas (representações sem interseções) a todo o instante para poderem ser representadas em Bases de Dados Espácio-temporais. Para obter representações realistas é necessário recorrer a técnicas de interpolação que não deformem o objeto ao longo da transformação, mantendo as suas propriedades físicas. Existem dois estudos principais, realizados por *Tøssebro et al* [7] e, mais recentemente, por *Mckennney et al* [8], focados na criação de representações espácio-temporais contínuas a partir de uma sequência de imagens capturadas em momentos discretos. No entanto estes focam-se apenas em técnicas de interpolação capazes de conseguir obter transformações topologicamente válidas ao longo de toda a transformação, não considerando as possíveis deformações que ocorrem e que podem resultar em representações pouco realistas do objeto móvel ao longo do tempo.

Recentemente foram propostas duas abordagens para criar representações contínuas do movimento de translação, rotação e deformação de objetos móveis a partir de observações em intervalos de tempo discretos [9] [10]. A primeira centra-se no *Vertex Correspondence Problem* e no desenvolvimento do suporte para a inserção de objetos móveis em Bases de Dados Espácio-temporais. Na segunda a ênfase foi o desenvolvimento de ferramentas para a aquisição das representações geométricas a partir de sequências de imagens e no *Vertex Path Problem*, de modo a ser possível obter representações fidedignas da evolução de um objeto móvel.

1.2 OBJETIVOS

Esta Dissertação centra-se no *Vertex Path Problem* e na exploração de técnicas de *Morphing* para representar a transformação de objetos móveis ao longo do tempo em bases de dados. O principal objetivo é investigar soluções que permitam obter transformações mais realistas do

que as obtidas usando os métodos propostos na literatura em bases de dados espácio-temporais.

O problema da transformação de um objeto noutro objeto já foi amplamente estudado e já existem soluções interessantes que podem ser aplicadas em vários contextos, por exemplo, na transformação de imagens 2D, polígonos, curvas ou representações volumétricas. A principal questão que se coloca, e que será explorada nesta Dissertação, consiste em avaliar se estas soluções podem ser adaptadas para representar objetos móveis em bases de dados espácio-temporais ou SIG, nomeadamente:

- Obter transformações realistas: uma vez que se pretende representar objetos com propriedades físicas, como o tamanho, a forma, a posição ou a orientação destes ao longo do tempo, é importante ser capaz de estimar transformações geométricas como translação, rotação ou deformação que representem o comportamento dos objetos o mais aproximadamente possível. A ênfase deste trabalho centra-se no estudo de métodos que permitam representar a evolução da forma dos objetos, evitando deformações irrealistas.

- A topologia dos objetos deve ser válida em todos os instantes durante uma transformação, ou seja, a representação da transformação dos objetos deve ser compatível com as restrições topológicas impostas à representação de objetos em bases de dados espácio-temporais ou SIG. Por exemplo, as arestas que definem as fronteiras de um polígono não devem intersetar-se durante uma transformação.

- A solução deve ser simples e computacionalmente pouco exigente, tendo em vista a implementação de operações espácio-temporais complexas, por exemplo, projeções, operações topológicas ou cálculo de distâncias, de forma eficiente, mesmo para volumes de dados significativos.

Para atingir estes objetivos, é explorada uma técnica de *Morphing* baseada em Triangulação Compatível de duas geometrias que representam um objeto móvel em dois momentos distintos. Desta forma, a transformação de um objeto com uma geometria eventualmente complexa pode ser representada através de um conjunto de operações mais simples, correspondendo à transformação individual de cada um dos seus triângulos. Este conjunto de transformações pode ser representado usando uma matriz e operações matriciais simples, o que resulta em operações computacionalmente pouco exigentes, satisfazendo o último requisito listado acima. No entanto, é ainda necessário avaliar esta técnica em relação aos outros dois requisitos.

O trabalho realizado nesta Dissertação consiste em duas componentes principais. A primeira centra-se na implementação de uma técnica de Triangulação Compatível, tendo em vista a decomposição de duas geometrias (polígonos) em malhas com o mesmo número de triângulos, e na implementação de técnicas de suavização, tendo em vista a melhoria da qualidade das malhas de triângulos. A segunda componente centra-se na aplicação de técnicas

de *Morphing* sobre as malhas de triângulos para representar a transformação dos objetos móveis ao longo do tempo. Para tal, foram implementadas diferentes técnicas de interpolação, tendo em vista a obtenção de transformações realistas e topologicamente válidas, que são comparadas com os resultados obtidos em trabalhos anteriores baseados em Interpolação Linear. Foi desenvolvida uma interface gráfica para executar os algoritmos implementados neste trabalho. Esta aplicação tem fins exclusivamente demonstrativos.

Os resultados deste trabalho são avaliados através de dados sintéticos e dados reais. Os primeiros são usados para testar as principais características dos métodos implementados e para pôr em evidência propriedades ou problemas de cada uma das soluções implementadas. Os segundos são usados para avaliar se as transformações obtidas usando os diversos métodos são realistas.

1.3 ORGANIZAÇÃO DO DOCUMENTO

A Dissertação está organizada da seguinte forma:

- **Capítulo 1 - Introdução**: é apresentado um enquadramento do problema num contexto global e quais os objetivos propostos nesta Dissertação.
- **Capítulo 2 - Representação e Transformação de Polígonos**: apresenta o estado da arte relativamente aos processos de Triangulação e *Morphing*.
- **Capítulo 3 - Técnicas de *Morphing* com Triangulação Compatível**: descreve as implementações das ferramentas desenvolvidas para obter transformações realistas de objetos móveis.
- **Capítulo 4 - Resultados**: apresenta os resultados práticos das ferramentas desenvolvidas para dois conjuntos de dados: um conjunto sintético com polígonos preparados para os testes e um conjunto real, onde é utilizada a solução implementada para representar o movimento de *Icebergs* ao longo do tempo.
- **Capítulo 5 - Considerações Finais**: descreve as conclusões desta Dissertação, bem como o trabalho futuro para que os resultados atuais possam ser melhorados.

capítulo 2

Representação e Transformação de Polígonos

2.1 introdução

Um dos principais problemas na representação objetos espaciais consiste em estimar a sua transformação ao longo do tempo. Este é um problema bem conhecido na área de *Morphing*, sendo denominado *Vertex Path Problem*. Uma vez que os objetos têm propriedades físicas, é importante evitar deformações e interseções ao longo da transformação. Para isso, o *Morphing* pode ser combinado com técnicas de Triangulação, onde os polígonos vão ser subdivididos em vários triângulos para que, em vez que efetuar uma transformação global do polígono, sejam efetuadas transformações individuais dos vários triângulos.
Neste capítulo são apresentadas as principais técnicas relativas ao trabalho desenvolvido nesta Dissertação, mais concretamente Triangulação Compatível e *Morphing*, assim como o trabalho existente sobre as mesmas.

A Secção 2.2 apresenta uma introdução sobre Triangulação, onde é explicado em que consiste uma Triangulação e as suas fases, sendo de seguida abordadas técnicas de Triangulação existentes.

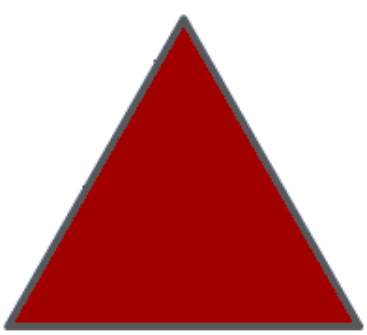
A Secção 2.3 descreve o conceito de *Morphing*, onde é feita uma introdução da técnica e abordadas diferentes soluções para o *Vertex Path Problem*.

A Secção 2.4 apresenta um resumo dos tópicos abordados ao longo do capítulo.

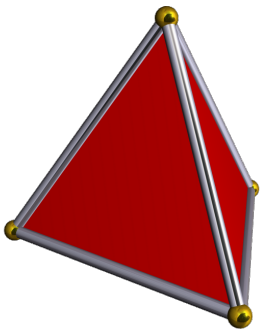
2.2 TRIANGULAÇÃO

2.2.1 CONTEXTUALIZAÇÃO

A triangulação de um polígono é uma técnica que consiste em fazer uma subdivisão do mesmo em vários elementos denominados *simplexs*, sendo um *simplex* o polígono mais simples da sua dimensão. Relativamente a duas dimensões, também conhecidos como *2-simplex*, o polígono mais simples é aquele com menos vértices e arestas, ou seja, triângulos (Figura 2.1a). Em três dimensões, também conhecidos como *3-simplex*, o polígono mais simples é o que tem menos vértices, arestas e faces, ou seja, tetraedros (Figura 2.1b).



(a) 2-simplex: Triângulo



(b) 3-simplex: Tetraedro

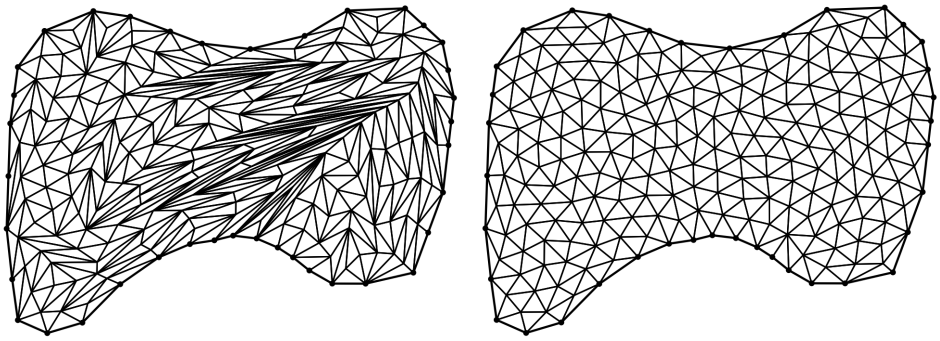
Figura 2.1: Exemplos de simplexos, Fonte: [11]

Qualquer método de triangulação é constituído por duas fases:

- Geração da malha de triângulos:
  - Consiste na subdivisão do polígono em vários triângulos, podendo cada um destes ter várias formas.
- Melhoria da malha de triângulos:
  - Suavização: É a técnica mais conhecida e utilizada nas melhorias das malhas de triângulos, sendo também conhecida como *Smoothing*. Consiste em melhorar a malha através do deslocamento dos vértices interiores, conhecidos como Vértices de Steiner, preservando a estrutura da malha. Um dos algoritmos mais conhecidos para esta técnica é o *Laplacian Smoothing* [12] que escolhe a nova posição dos vértices tendo em conta a posição dos seus vizinhos, no entanto este algoritmo gera muitas vezes malhas inválidas. Outras técnicas de Suavização foram estudadas por *Xu et al* [13], *Zhou et al* [14] e *Surazhsky et al* [15], técnicas estas que escolhem a nova posição dos vértices tendo em conta uma melhoria dos ângulos de cada triângulo, aumentando assim os ângulos mínimos da malha de triângulos.

– *Remeshing*: É uma técnica que melhora a malha de triângulos através de alterações na conectividade entre os vértices, tendo esta técnica já sido estudada por *Surazhsky et al* [15], com o objetivo de maximizar as áreas mínimas e por *Ebene et al* [16], onde o foco é a maximização dos ângulos mínimos.

As diferenças entre uma malha normal e uma malha melhorada podem ser verificados na Figura 2.2.



(a) Malha Original (b) Malha melhorada

Figura 2.2: Diferença entre malha normal e malha melhorada, Fonte: [15]

2.2.2 TRIANGULAÇÃO DE DELAUNAY

A técnica de triangulação mais conhecida é a Triangulação de Delaunay [17], tendo sido inventada em 1934 por Boris Delaunay. Esta é uma técnica cujo objetivo é a maximização dos ângulos mínimos, de modo a que sejam evitados triângulos longos e finos na malha de triângulos final.

Um conceito importante deste algoritmo é o *Edge Flip*, que consiste em fazer uma troca da aresta de ligação dos vértices, sendo este utilizado, por exemplo, quando é detectado que existe uma ligação que não maximiza os ângulos mínimos. Esta situação está representada na Figura 2.3, onde o polígono tem duas ligações possíveis. A ligação efetuada na imagem do lado esquerdo não maximiza os ângulos mínimos e, como tal, é considerada inválida. De modo a tornar a ligação válida é feito um *Edge Flip*, ou seja, neste caso trocou a ligação dos vértices $\bar{p}_i\bar{p}_j$ pela ligação $\bar{p}_k\bar{p}_l$, que é a ligação que maximiza o ângulo mínimo.

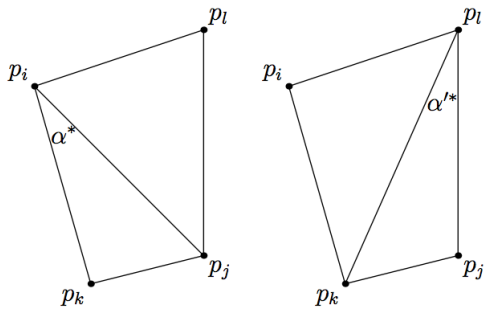
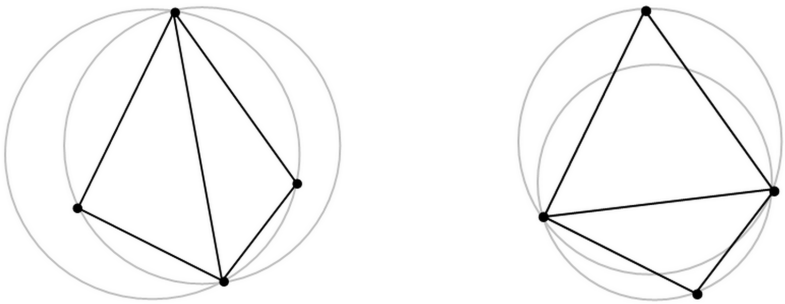


Figura 2.3: Exemplo de Edge Flipping, Fonte: [18]

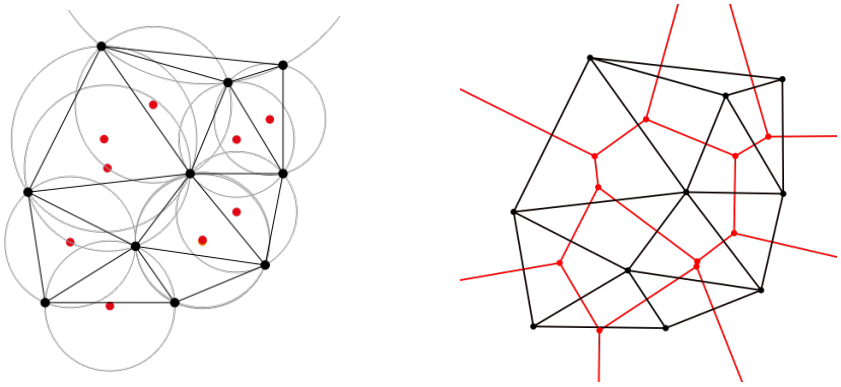
Existe outro conceito importante numa Triangulação de Delaunay que é o Critério do Círculo Circunscrito Vazio, também conhecido como Critério de Delaunay. Este critério é utilizado para verificar a validade da malha de triângulos, onde cada triângulo têm um circulo associado aos seus vértices, circulo este que não pode conter nenhum vértice no seu interior, caso contrário a triangulação é inválida, como exemplificado na Figura 2.4a. Para corrigir a situação é necessário aplicar um *Edge Flip*, tal como exemplificado na Figura 2.4b.



(a) Triangulação de Delaunay inválida (b) Triangulação de Delaunay válida

Figura 2.4: Exemplo de malha inválida e a sua correção, Fonte: [19]

A Figura 2.5a mostra um exemplo de uma Triangulação de Delaunay finalizada, em que é possível verificar que não existe nenhum vértice no interior de qualquer circulo, apenas nas suas extremidades, pelo que esta triangulação é válida. Existe ainda uma relação entre a Triangulação de Delaunay e o Diagrama de Voronoi, onde este último pode ser obtido através da interligação dos vários centros dos círculos circunscritos que conectam os triângulos na Triangulação de Delaunay, situação demonstrada na Figura 2.5b.



(a) Triangulação Delaunay (b) Diagrama de Voronoi

Figura 2.5: Relação entre a Triangulação Delaunay e o Diagrama de Voronoi, Fonte: [19]

Em termos práticos, a implementação desta técnica de triangulação foi estudada, por exemplo, por *Lee et al* [20] que propôs dois algoritmos. O primeiro resulta da utilização de uma estratégia iterativa, enquanto que o segundo recorre a uma estratégia Dividir e Conquistar, ou seja, uma solução recursiva, tendo esta última apresentado melhores desempenhos de execução quando comparada com a solução iterativa.

No entanto a Triangulação de Delaunay têm um problema quando utilizada para triangular dois polígonos com correspondências entre si, uma vez que é muito provável que o número de triângulos de ambas as malhas resultantes seja diferente, o que é um fator indesejável. Na subsecção seguinte é abordada uma nova técnica de triangulação para a resolução deste problema.

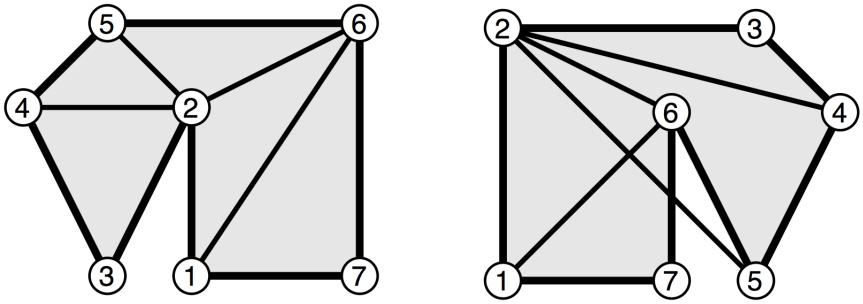
2.2.3 TRIANGULAÇÃO COMPATÍVEL

Triangulação Compatível é uma técnica utilizada para gerar uma malha de triângulos compatível entre dois polígonos, origem e destino, de forma a que, no final, as malhas resultantes tenham exatamente o mesmo número de triângulos.

O termo compatível refere-se ao interior dos polígonos, ou seja, à malha de triângulos obtida. Para esta ser compatível, existem algumas condições que necessitam de ser verdadeiras:

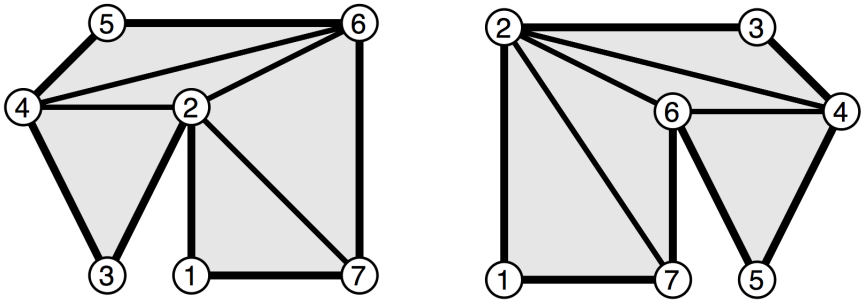
- o número de triângulos gerados na triangulação do polígono origem tem de ser o mesmo gerado na triangulação do polígono destino;
- as ligações vértice-a-vértice existentes em cada triângulo não ultrapassam a fronteira do polígono;
- as ligações vértice-a-vértice existentes em cada triângulo não intercetam outras ligações.

Se alguma das condições referidas acima não for verdadeira, então a triangulação é incompatível. As Figuras 2.6 e 2.7 demonstram, respetivamente, um exemplo de triangulação incompatível e compatível para o mesmo conjunto de vértices, onde a correspondência dos vértices é feita pelo identificador numérico dos mesmos.



(a) Polígono Origem (b) Polígono Destino

Figura 2.6: Exemplo de Triangulação incompatível, Fonte: [15]



(a) Polígono Origem (b) Polígono Destino

Figura 2.7: Exemplo de Triangulação compatível, Fonte: [15]

Como podemos verificar na Figura 2.6, a triangulação é incompatível porque a ligação entre os vértices 2-5, no polígono destino (Figura 2.6b), além de ultrapassar as fronteiras também interceta outras ligações existentes, neste caso as ligações entre os vértices 1-6 e 6-7. Esta situação é resolvida através de alterações na ligação entre vértices, apresentadas na Figura 2.7.

É importante referir que qualquer técnica de Triangulação Compatível requer que primeiro seja encontrada uma correspondência de vértices entre os polígonos origem e destino, sendo este problema conhecido como *Vertex Correspondence Problem*, já estudado nas Dissertações do Luís Paulo [9] e Pedro Mesquita [10].

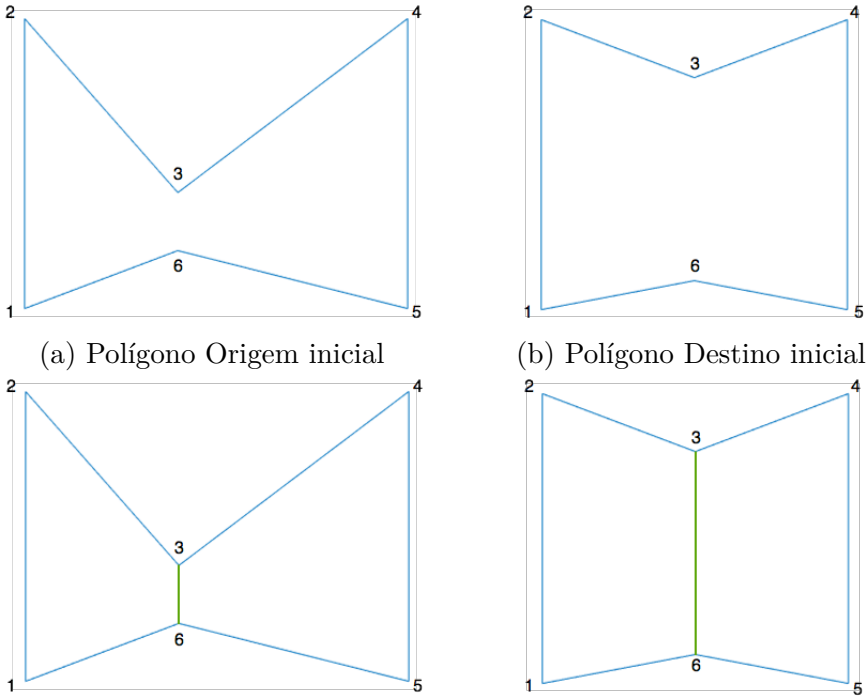
Existem propostas que implementam a técnica de triangulação descrita acima:

Boris Aronov et al [21] propôs uma das primeiras técnicas para gerar Triangulações Compatíveis entre dois polígonos com recurso à adição de Vértices de Steiner, no entanto esta proposta é muito complexa e pouco eficiente, uma vez que implica a adição de um número elevado de Vértices de Steiner de modo a obter os resultados pretendidos.

Kranakis et al [22] propôs duas implementações. A primeira implementação acaba por introduzir um número maior de vértices de Steiner, o que aumenta a sua complexidade. A segunda implementação já teve uma melhoria na complexidade do algoritmo, uma vez que introduz um número de Vértices de Steiner mais baixo comparado com a anterior, no entanto estes Vértices de Steiner adicionados podem ser colocados nas fronteiras do polígono. Em ambos os métodos, o número de vértices de Steiner introduzidos depende do número de vértices reflexos, que são vértices onde o ângulo interno formado pelas suas arestas é maior que 180^{o}, presentes em cada polígono.

Surazhsky et al [15] apresenta um algoritmo que teve como base o estudo feito anteriormente por *Gupta et al* [23], que propôs um algoritmo com base em partições de polígonos através de ligações de custo mínimo. A sua proposta é similar, pois também é baseado em ligações de vértices mais próximos entre si, no entanto o seu conceito geral é mais simples e apenas introduz Vértices de Steiner quando necessário. Esta técnica foi a que utilizei para a geração de Triangulações Compatíveis, sendo descrita com maior detalhe em seguida.

O primeiro passo consiste em analisar ambos os polígonos para obter o par de vértices, não vizinhos, mais próximos entre si (Figura 2.8).



(a) Polígono Origem inicial (b) Polígono Destino inicial

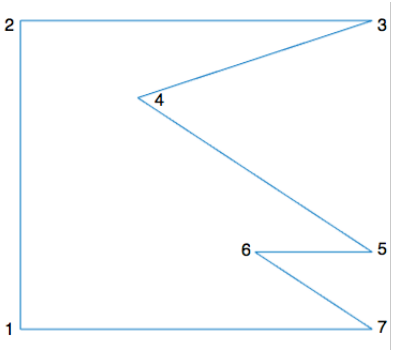
(c) Polígono Origem depois da ligação (d) Polígono Destino depois da ligação

Figura 2.8: Exemplo de ligação do par de vértices

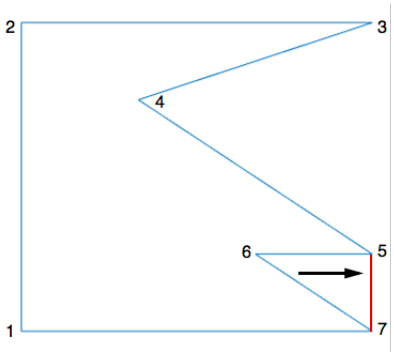
Analisando ambos os polígonos, verifica-se que os vértices 3-6 do polígono origem (Figura 2.8a) são o par com menor distância entre si, ou seja, vai ser efetuada uma ligação entre eles em ambos os polígonos como é exemplificado nas Figuras 2.8c e 2.8d.

No entanto, existem duas situações especiais a ter em conta:

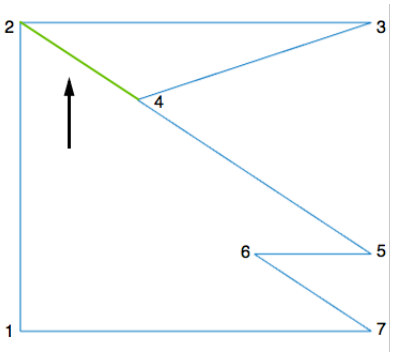
- Ligação do par de vértices pode não ser válida, por intercetar ligações já existentes ou por ultrapassar as fronteiras do polígono, tal como exemplificado na Figura 2.9.
- Ligação correspondente no outro polígono pode não ser válida, pelas mesmas razões descritas anteriormente, tal como exemplificado na Figura 2.10.



(a) Polígono Original



(b) Ligação Inválida

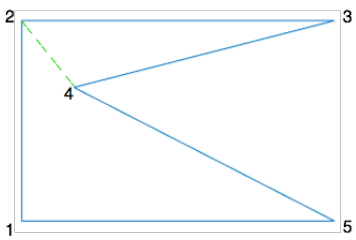


(c) Ligação Válida

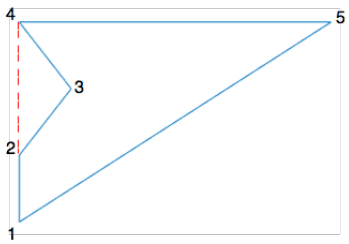
Figura 2.9: Validação da ligação de vértices

Considerando o polígono original, representado na Figura 2.9a, verifica-se que o par de vértices com menor distância entre são si são os vértices 5 e 7, no entanto a ligação entre estes dois vértices não é possível porque ultrapassa as fronteiras do polígono (Figura 2.9b). Por ser uma ligação inválida, este par de vértices é ignorado, continuando à procura até encontrar o par mais próximo com ligação válida sendo, neste caso, os vértices 2 e 4 (Figura 2.9c).

No caso em que a ligação correspondente no outro polígono não é válida entra um conceito importante: Vértices de Steiner, também conhecidos como Pontos de Steiner. Estes vértices, que não pertencem ao conjunto inicial do polígono, são adicionados automaticamente pelo algoritmo quando necessários de forma a possibilitar a compatibilidade da triangulação.

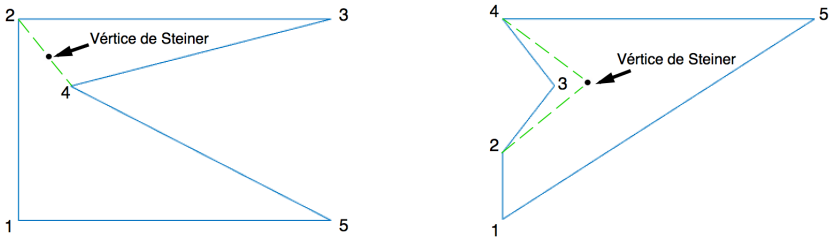


(a) Polígono Origem



(b) Polígono Destino

Figura 2.10: Exemplo onde é necessário Vértice de Steiner

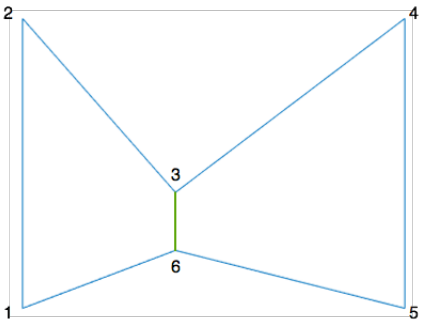


(a) Polígono Origem (b) Polígono Destino

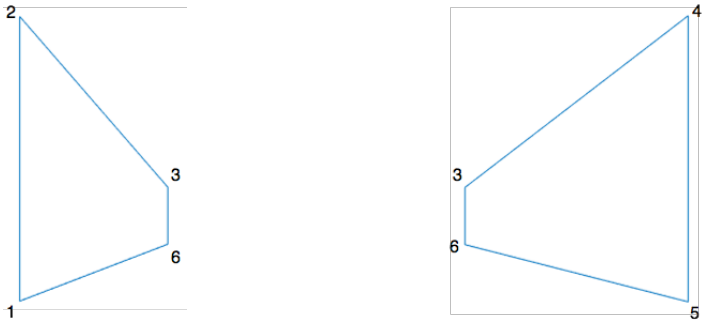
Figura 2.11: Resultado com a aplicação do Vértice de Steiner

As Figuras 2.10 e 2.11 demonstram, respetivamente, um exemplo onde é necessário um Vértice de Steiner e o resultado após a sua aplicação. Como o par de vértices mais próximos entre si são o 2-4 do polígono origem (Figura 2.10a), então vai ser efetuada uma ligação entre eles em ambos os polígonos, situação que é possível na Figura 2.10a, mas que é impossível na Figura 2.10b, uma vez que essa ligação ultrapassa as fronteiras do polígono destino. Para contornar esta situação e formar uma ligação válida é adicionado um Vértice de Steiner, tal como exemplificado na Figura 2.11.

Depois de encontrado o par de vértices, não vizinhos, mais próximos entre si e que formem uma ligação válida, é necessário fazer a geração dos dois novos sub-polígonos que essa ligação de vértices vai criar.

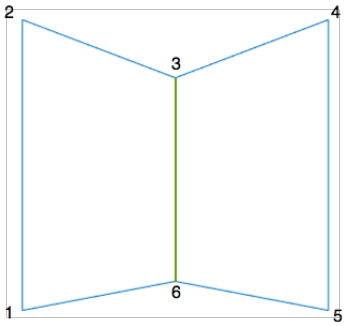


(a) Ligação no polígono origem

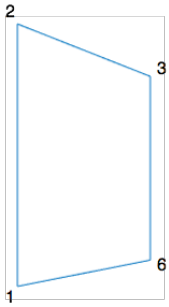


(b) 1º polígono gerado (c) 2º polígono gerado

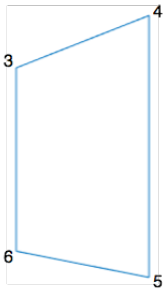
Figura 2.12: Geração dos sub-polígonos para polígono Origem



(a) Ligação no polígono destino



(b) 1º polígono gerado



(c) 2º polígono gerado

Figura 2.13: Geração dos sub-polígonos para polígono Destino

Nas Figuras 2.12 e 2.13 estão exemplificados os resultados da geração de sub-polígonos para a ligação do par de vértices representados em 2.12a e 2.13a.

Depois de efetuada esta geração, é feita uma chamada recursiva para todos os sub-polígonos com mais de três vértices, terminando quando todos ficarem com três vértices, ou seja, todos os sub-polígonos gerados são triângulos. Um exemplo, passo a passo (cada passo descrito em cada coluna), da aplicação do algoritmo pode ser observado na Figura 2.14. Neste exemplo, a primeira linha representa as várias ligações que foram efetuadas no polígono origem e a segunda linha representa as várias ligações que foram efetuadas no polígono destino durante a triangulação. É ainda possível verificar a aplicação de Vértices de Steiner quando são efetuadas as ligações entre os vértices 1-4 (coluna 4) e 5-8 (coluna 8).

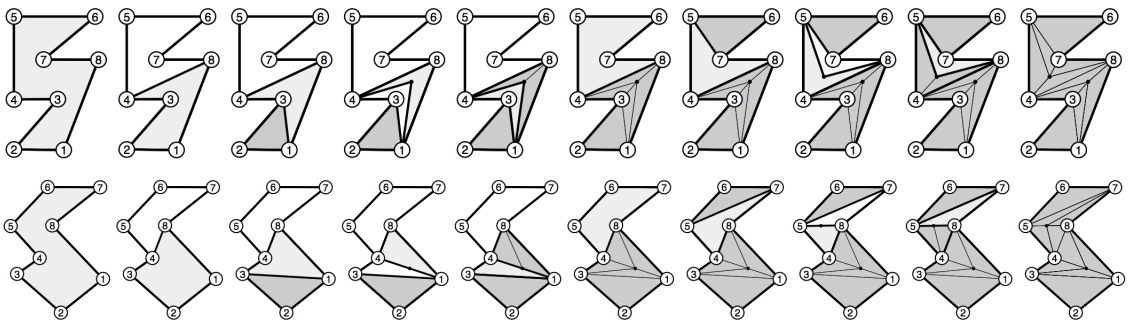


Figura 2.14: Exemplo de triangulação compatível, Fonte: [15]

A Triangulação Compatível de dois polígonos tem uma possibilidade de aplicação bastante interessante, nomeadamente no *Morphing*, uma vez que com esta é possível fazer várias transformações independentes dos vários triângulos em vez de uma transformação do polígono global. Este tipo de aplicação traz vantagens, nomeadamente a possibilidade de utilização de técnicas de Interpolação Rígida, abordadas na secção seguinte, que têm como objetivo preservar as propriedades físicas do objeto ao longo da transformação.

2.3 MORPHING

2.3.1 CONTEXTUALIZAÇÃO

Morphing é uma técnica utilizada para fazer uma transformação gradual entre os polígonos origem e destino. Esta técnica tem sido muito investigada ao longo dos anos, onde o principal desafio tem sido centrado no desenvolvimento de algoritmos que consigam efetuar um bom *Morphing*, sendo que um bom *Morphing* deve:

- efetuar uma transformação natural que preserva as propriedades dos objetos, evitando deformações;
- Gerar uma transformação sem interseções.

Figura 2.15: Exemplo de um bom *Morphing* entre um elefante e uma girafa, Fonte: [6]

Num processo de *Morphing* de dois polígonos podem ser utilizadas até quatro tipos de transformações:

- Rotação;
- Mudança de Escala;
- *Shear*: transformação geométrica que modifica a forma do objeto;
- Translação.

Através das transformações indicadas acima é possível efetuar qualquer *Morphing* entre dois polígonos, sendo que aqui entra um conceito importante para esta técnica, denominada de matriz afina. Esta é a matriz que contém as componentes de Rotação, Mudança de Escala

e *Shear* da transformação, sendo utilizada para fazer a transformação do polígono origem no destino onde, separadamente, é efetuada a Translação porque esta apenas indica a posição final do polígono.

Tendo em conta que nesta secção será abordado o *Morphing* de triângulos, considera-se que cada triângulo está representado por uma matriz de dimensão 2x3 e a matriz afina 2x2. A transformação do triângulo origem no destino consiste numa multiplicação da matriz afina pelas coordenadas do triângulo origem, que vai aplicar as componentes de Rotação, Mudança de Escala e *Shear*, resultando assim no novo objeto. Por último vai ser somada a componente de Translação para coloca-lo na posição correta. Este processo está representado na Equação 2.1, onde A representa a matriz afina, S o polígono origem, L a componente de Translação e T o polígono destino:

$$\begin{aligned} T &= A * S + L \\ \Leftrightarrow T &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} s_{x1} & s_{x2} & s_{x3} \\ s_{y1} & s_{y2} & s_{y3} \end{bmatrix} + \begin{bmatrix} l_x & l_x & l_x \\ l_y & l_y & l_y \end{bmatrix} \end{aligned} \tag{2.1}$$

A matriz afina pode ainda ter outro formato, onde além de ter as componentes de Rotação, Mudança de Escala e *Shear* também vai estar incluída a Translação, tendo neste caso dimensão 3x3 (Equação 2.2). Quando utilizado este formato, a Equação 2.1 fica ligeiramente diferente, sendo necessário acrescentar uma linha de 1's à matriz que contém as coordenadas do triângulo origem para a operação ser válida. As diferenças referidas estão representadas na Equação 2.3.

$$A = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$\begin{aligned} T &= A * S \\ \Leftrightarrow T &= \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} s_{x1} & s_{x2} & s_{x3} \\ s_{y1} & s_{y2} & s_{y3} \\ 1 & 1 & 1 \end{bmatrix} \end{aligned} \tag{2.3}$$

É importante referir que, tal como na Triangulação Compatível, o *Morphing* também requer que exista uma correspondência de vértices entre os polígonos origem e destino. Existindo esta correspondência, o principal desafio passa por encontrar uma solução para o *Vertex Path Problem*, problema este que é descrito na subsecção seguinte.

2.3.2 VERTEX PATH PROBLEM

É a fase mais delicada no processo de *Morphing*, uma vez que é nesta fase que se define como a transformação vai decorrer. A solução mais simples é obtida através de uma Interpolação Linear dos vértices, que é a técnica mais conhecida. Esta foi a técnica utilizada na Dissertação [9] e consiste em considerar que os trajetos dos vários vértices são linhas rectas e que estes se deslocam com uma velocidade constante. A Figura 2.16 mostra um exemplo de um triângulo sobre o qual é aplicada uma rotação de 180º, permitindo observar ainda, através do esquema de cores, o trajeto (linear) que cada um dos vértices faz até ao seu destino.

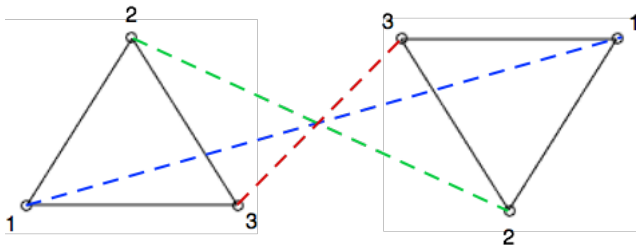


Figura 2.16: Conceito geral da interpolação linear

Uma Interpolação Linear é simples de implementar, bastando aplicar a Equação 2.1 ou 2.3 (mediante o formato da matriz afina) e fazê-la variar num espaço temporal t (Equação 2.4 ou 2.5, respetivamente), onde as componentes que variam com o tempo são a matriz afina e a Translação. A variável t pode tomar qualquer valor dentro do intervalo [0 ; 1], onde 0 é o instante inicial que corresponde ao polígono origem e 1 o instante final que corresponde ao polígono destino. Os outros instantes representam as formas intermédias do polígono.

$$T(t) = A(t) * S + L(t) \tag{2.4}$$

$$T(t) = A(t) * S \tag{2.5}$$

Esta abordagem, embora simples, não é apropriada, principalmente nos casos em que se faz o *Morphing* de objetos com propriedades físicas, pois os resultados intermédios não serão os desejados. O motivo disso é que, com exceção de transformações muito básicas, quase sempre são geradas interseções e/ou deformações do objeto, resultando numa transformação forçada e nada natural. Um exemplo desta situação é apresentado na Figura 2.17 onde é possível ver as diferenças com o mesmo exemplo anteriormente demonstrado na Figura 2.15.

Figura 2.17: *Morphing* com interpolação linear, Fonte: [6]

Assim sendo, é necessário abordar alternativas à Interpolação Linear para obter melhores resultados, existindo vários estudos de outros métodos de interpolação:

Na Dissertação [10] são estudadas duas soluções: a primeira solução recorre ao método proposto por *McKenney et al* [24] que apresenta uma técnica de *Morphing* onde garante que são sempre geradas regiões válidas, onde uma região válida é equivalente a um polígono sem interseções. A segunda solução faz uma divisão do movimento do objeto em Translação, Rotação e Deformação através de um alinhamento entre os polígonos origem e destino.

Alexa et al [6] e *Baxter et al* [25] propuseram técnicas de Interpolação Rígida em malhas de triângulos. Uma Interpolação Rígida é uma técnica utilizada para gerar transformações com o objetivo de preservar as suas propriedades físicas e evitar deformações do objeto. Nestas técnicas, é gerada uma matriz de transformação que procura fazer a transformação ideal, também conhecida como transformação ótima, de cada triângulo. As duas técnicas propostas têm pequenas diferenças mas os resultados são equivalentes. As diferenças entre as duas técnicas é no cálculo das posições dos vértices partilhados, onde na primeira a posição final destes é calculada através da minimização do erro quadrático, enquanto que na segunda é feita com recurso a equações normais.

Shapira et al [26] propôs uma técnica onde é introduzido um novo conceito para a representação dos polígonos, ao qual chamaram de representação *Star-Skeleton*. Esta representação é composta por duas componentes: uma decomposição que vai criar várias peças em forma de estrela e um esqueleto que interliga todas essas peças. A utilização desta técnica no *Morphing* de polígonos é feita através de uma interpolação das suas representações *Star-Skeleton*, em vez da interpolação dos seus vértices.

Sederberg et al [27] propôs uma técnica que é uma melhoria da Interpolação Linear, já referida anteriormente. Nesta técnica, os fatores em análise de cada polígono são o comprimento das suas arestas e os seus ângulos, sendo estes os fatores que são interpolados, gerando uma melhor transformação, ao contrário da Interpolação Linear que faz uma interpolação da posição dos vértices.

Xu et al [28] propôs uma técnica com o objetivo de fazer uma Interpolação Rígida, no entanto esta técnica acaba por ser similar à de *Alexa et al* [6], já referida anteriormente, com a diferença que faz uma interpolação através de equações de *Poisson* e que usa as áreas dos triângulos como peso nessas interpolações

Gotsman et al [29] propôs uma técnica de *Morphing* com Triangulação Compatível, onde o grande objetivo é garantir que não existem interseções nas transformações intermédias. O ponto principal desta técnica é que, além de ser efetuada uma Triangulação Compatível entre

os dois polígonos, é efetuada uma segunda Triangulação Compatível dos polígonos com um polígono externo, que é conhecida como *Annuli Compatible Triangulation*. Em seguida são geradas as respetivas interpolações, onde em vez de serem interpoladas as coordenadas dos vértices são interpoladas as coordenadas dos seus baricentros.

No âmbito deste trabalho, as técnicas mais interessantes são propostas por *Gotsman et al* [29], *Alexa et al* [6] e *Baxter et al* [25], sendo as duas últimas as que são utilizadas neste trabalho, pelo facto de serem técnicas de Interpolação Rígida onde o seu principal objeto é preservar as propriedades físicas dos objetos durante a transformação.

2.3.3 INTERPOLAÇÃO RÍGIDA

Interpolação Rígida é uma técnica onde o principal objetivo é preservar as propriedades físicas do objeto à medida que é feita a transformação do polígono origem para o destino, ou seja, efetuar uma transformação com o mínimo de deformação possível.
Anteriormente foi demonstrado que a abordagem básica para efetuar uma transformação consiste em utilizar uma Interpolação Linear, através da Equação 2.4. No entanto, as propostas de *Alexa et al* [6] e *Baxter et al* [25] consistem em alterar a forma como essa transformação é efetuada. Utilizaram o estudo realizado por *Shoemake et al* [30], onde em vez de interpolar diretamente a matriz afina, que já contém todas as transformações a fazer, esta vai ser decomposta num produto de matrizes. Foram utilizados vários tipos de decomposição, no entanto os melhores resultados foram obtidos através do *Singular Value Decomposition* (SVD) [31], através do qual foi possível fazer uma interpolação da Rotação e da Deformação (*Shear* e Mudança de Escala) em separado, criando assim uma transformação ótima, sendo que o termo ótimo refere-se à transformação com menor deformação do objeto.

Nesta fase é utilizada a matriz afina com as dimensão 2x2, que contém as componentes de Rotação, Mudança de Escala e *Shear*, sendo a Translação calculada separadamente no fim da transformação. Após a aplicação do SVD, a matriz afina vai ser decomposta num produto de três matrizes: R_α, D e R_β (Equação 2.6). Estas matrizes representam, respetivamente, uma sequência de três transformações: uma Rotação, uma Mudança de Escala e outra Rotação. Através de operações matemáticas, obtiveram uma nova decomposição que representa a transformação através de um produto de duas matrizes, onde uma corresponde a uma matriz de Rotação e outra à matriz de Deformação, que contém a Mudança de Escala e o *Shear* (Equação 2.7). Estas transformações vão ser interpoladas em separado, de modo a obter a matriz que efetua a transformação ótima, A_γ, referida anteriormente.

$$A = R_\alpha * D * R_\beta \tag{2.6}$$
$$\Leftrightarrow A = R_\alpha * (R_\beta * R_\beta^T) * D * R_\beta$$
$$\Leftrightarrow A = (R_\alpha * R_\beta) * (R_\beta^T * D * R_\beta)$$
$$\Leftrightarrow A_\gamma = R_\gamma * S \tag{2.7}$$

A matriz de transformação ótima, A_γ, é utilizada para definir a transformação, seguindo a mesma abordagem usada para a Interpolação Linear (Equação 2.4). Para isso é necessário fazer a matriz de transformação ótima variar no espaço temporal t. Depois de obtida essa variação é possível calcular qualquer forma intermédia da transformação de um polígono origem, S, no destino, T, onde o espaço temporal t pode tomar qualquer valor entre [0 ; 1] (Equação 2.8).

$$T(t) = A_\gamma(t) * S + L(t) \tag{2.8}$$

A Figura 2.18 exemplifica a transformação de um triângulo com os dois tipos de interpolação abordados: Interpolação Linear e Interpolação Rígida. É possível verificar, através das formas intermédias, que na Interpolação Rígida vai existindo uma menor variação da área dos triângulos comparativamente com a Interpolação Linear, o que se traduz numa transformação mais natural. Na última coluna é ainda possível ver o trajeto que os vértices fizeram até chegar ao destino, onde podemos verificar que na Interpolação Linear os vértices fazem um trajeto direto até ao destino, enquanto na Interpolação Rígida fazem um trajeto com rotação.

(a) Interpolação Linear

(b) Interpolação Rígida

Figura 2.18: Diferenças entre Interpolação Linear e Rígida, Fonte: [6]

Até agora tem sido abordada a transformação ótima de apenas um triângulo, no entanto quando estamos a lidar com uma malha de triângulos o cenário é um pouco diferente, uma vez que existem vértices partilhados por mais que um triângulo e as transformações ideais desses triângulos podem gerar posições diferentes para o mesmo vértice partilhado, situação esta que é abordada na subsecção seguinte.

2.3.4 POSICIONAMENTO DOS VÉRTICES PARTILHADOS

Este problema também é abordado por *Alexa et al* [6] e *Baxter et al* [25], onde são propostas duas soluções distintas, mas com resultados equivalentes. Na primeira é apresentada uma solução com recurso à minimização direta do erro quadrático enquanto na segunda a solução apresentada é com base em equações normais, sendo esta última a solução que vou abordar nesta subsecção, dado a sua maior simplicidade de implementação.

Uma vez que, com uma transformação ótima, os vértices partilhados vão ter mais que uma coordenada, o objetivo é que exista apenas uma coordenada para cada vértice e que essa coordenada seja a mais próxima possível da transformação ótima inicialmente calculada, para qualquer instante da transformação, sendo que o instante 0 corresponde ao polígono origem e o instante 1 ao polígono destino.

$$
\begin{aligned}
&V(t) = (v_1(t), v_2(t), ..., v_n(t)), t \in [0,1] \wedge n = n^{\text{o}} vértices \\
&V(0) = (s_1, s_2, ..., s_n) \\
&V(1) = (t_1, t_2, ..., t_n)
\end{aligned}
$$

Na resolução deste problema, *Baxter et al.* [25] começaram por rescrever a matriz afina, que transforma um triângulo P_T num triângulo Q_T, onde T é o identificador do triângulo com índices numéricos (i, j, k), como sendo:

$$A_T = \hat{P}_T * Q_T \tag{2.9}$$

onde:

$$
P_T = \begin{bmatrix} p_i^x & p_i^y \\ p_j^x & p_j^y \\ p_k^x & p_k^y \end{bmatrix} Q_T = \begin{bmatrix} q_i^x & q_i^y \\ q_j^x & q_j^y \\ q_k^x & q_k^y \end{bmatrix} D = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}
$$

$$\hat{P}_T = (D * P_T)^{-1} * D \tag{2.10}$$

Esta reformulação permite utilizar o mesmo formato para definir uma segunda matriz de transformação, B, que vai fazer uma transformação do mesmo triângulo P para as coordenadas dos vértices no instante t, V(t), coordenadas estas que são desconhecidas.

$$B_T(t) = \hat{P}_T * V_T(t) \tag{2.11}$$

O objetivo desta segunda matriz de transformação é ser utilizada, na malha de triângulos, para encontrar a transformação que minimiza o erro entre B(t) e a transformação ótima A(t).

$$E(t) = \sum_{T=1}^{M} \| \hat{P}_T * V_T(t) - A_T(t) \|^2 \tag{2.12}$$

Neste formato, $V_T(t)$ contém as posições dos três vértices do triângulo no instante t da transformação. De modo a tornar esta matriz numa de dimensão Nx2, onde N=nºvértices do polígono, ou seja, numa matriz com as coordenadas de todos os vértices, $\bar{V}$, é necessário alterar a estrutura de $\hat{P}_T$ para uma matriz esparsa, $\bar{P}$, de dimensão 2xN. Nesta matriz, todos os elementos vão ser 0's, com exceção dos que se encontram nas colunas correspondentes aos índices dos vértices do triângulo representado por T, tendo estes os respetivos valores calculados para $\hat{P}_T$. Mais concretamente, por exemplo, no *Morphing* de um triângulo T, com os índices numéricos (i, j, k), todos os elementos vão ser 0's excepto as colunas i, j e k, como representado na Equação 2.14.

$$E(t) = \sum_{T=1}^{M} \|\bar{P}_T * \bar{V}(t) - A_T(t)\|^2 \tag{2.13}$$

$$\bar{P}_T = \begin{array}{c} \begin{matrix} \quad\mathsf{i} & \quad\mathsf{j} & \quad\mathsf{k} \end{matrix} \\ \begin{bmatrix} \cdots & [\hat{P}_T]_{11} & \cdots [\hat{P}_T]_{12} \cdots [\hat{P}_T]_{13} \cdots \\ \cdots & [\hat{P}_T]_{21} & \cdots [\hat{P}_T]_{22} \cdots [\hat{P}_T]_{23} \cdots \end{bmatrix} \end{array} \tag{2.14}$$

Assim já é possível expressar o problema sob a forma de equações normais (Equação 2.18) onde, através do Método dos Quadrados Mínimos (*Linear Least Squares*), é obtida a solução de erro mínimo que resolve a equação. Todas as componentes da equação são vectores com informação global de toda a malha de triângulos, como é possível visualizar nas Equações 2.15 e 2.16, onde M corresponde ao número de triângulos existente. É ainda possível ver toda a dedução da equação final em 2.17. No primeiro passo da dedução, uma vez que $\bar{P}$ não é uma matriz quadrada e como tal não pode ser invertida, cada uma das partes da equação é multiplicada por $\bar{P}^\intercal$, de modo a que no lado esquerdo da equação seja efetuada uma multiplicação entre $\bar{P}^\intercal$ e $\bar{P}$, que vai gerar uma matriz quadrada. Esta matriz já pode ser invertida, o que acontece no segundo passo, onde ambas as partes da equação são multiplicadas pela inversa da matriz quadrada. Esta situação gera, no lado esquerdo da equação, uma multiplicação da inversa com a normal, que vai resultar na matriz identidade, I. Como a multiplicação de uma matriz com a identidade resulta na própria matriz, então $I * \bar{V}(t) = \bar{V}(t)$, obtendo assim a solução da equação.

$$\bar{P} = \begin{bmatrix} \bar{P}_1^\intercal \cdots & \bar{P}_M^\intercal \end{bmatrix}^\intercal \tag{2.15}$$

$$\bar{A}(t) = \begin{bmatrix} A_\gamma(t)_1^\intercal \cdots & A_\gamma(t)_M^\intercal \end{bmatrix}^\intercal \tag{2.16}$$

$$\bar{P} * \bar{V}(t) = \bar{A}(t) \tag{2.17}$$

$$\Leftrightarrow \bar{P}^{\intercal} * \bar{P} * \bar{V}(t) = \bar{P}^{\intercal} * \bar{A}(t) \tag{2.18}$$

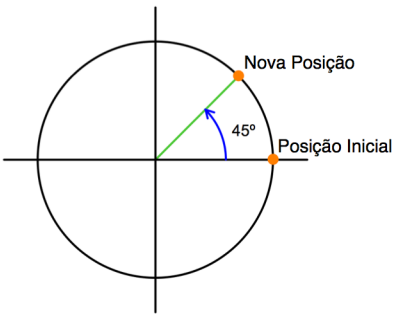
$$\Leftrightarrow (\bar{P}^{\intercal} * \bar{P})^{-1} * (\bar{P}^{\intercal} * \bar{P}) * \bar{V}(t) = (\bar{P}^{\intercal} * \bar{P})^{-1} * \bar{P}^{\intercal} * \bar{A}(t)$$

$$\Leftrightarrow I * \bar{V}(t) = (\bar{P}^{\intercal} * \bar{P})^{-1} * \bar{P}^{\intercal} * \bar{A}(t)$$

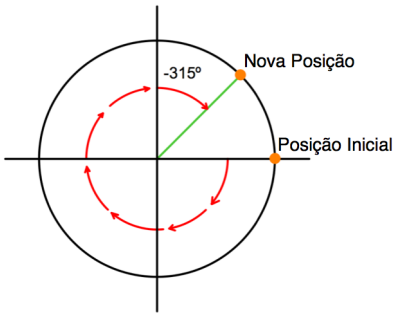
$$\Leftrightarrow \bar{V}(t) = (\bar{P}^{\intercal} * \bar{P})^{-1} * \bar{P}^{\intercal} * \bar{A}(t)$$

2.3.5 CONSISTÊNCIA DAS ROTAÇÕES

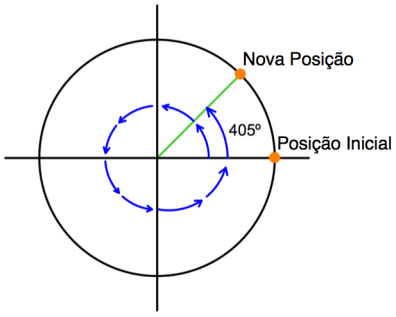
Nesta subsecção é abordado um problema recorrente nos processos de *Morphing*, associada à componente de Rotação da transformação e que está representada na Figura 2.19. O primeiro exemplo representa uma rotação de 45º em torno da origem, seguida de duas rotações diferentes, uma de -315º e outra de 405º, mas que no entanto produzem o mesmo resultado final. A situação referida pode fazer com que a componente de Rotação seja interpolada incorretamente, situação que acontece frequentemente, uma vez que numa matriz de rotação o ângulo encontra-se sempre entre [-180 ; 180], de modo a que seja sempre efetuada a rotação com menor magnitude, mesmo que esta não corresponda à melhor rotação global do objeto.



(a) Rotação de 45º



(b) Rotação de -315º



(c) Rotação de 405º

Figura 2.19: Exemplos de rotações com o mesmo resultado final

Estas inconsistências nas rotações são prejudiciais para o processo de *Morphing*, uma vez

que podem retirar a naturalidade da transformação, ou até mesmo gerar interseções. *Baxter et al* [25] propôs uma técnica com o objetivo de melhorar a consistência das componentes de rotação entre os vários triângulos, técnica esta que é composta por duas fases:

- Remoção de inconsistências;
- Minimização de rotações.

A primeira fase consiste em analisar as matrizes de rotação, $R_\gamma(t)$ da Equação 2.7, dos vários triângulos, de onde vão ser extraídos os ângulos de rotação para verificar se existem descontinuidades nas rotações entre triângulos vizinhos, sendo que é considerado que existe uma descontinuidade se a diferença entre os ângulos de rotação de triângulos vizinhos ultrapassarem os 180^o. Isto acontece quando os triângulos vizinhos têm sentidos de rotação contrários, que foram geradas para obter a rotação de menor magnitude. Nos casos em que existam descontinuidades, vão ser adicionados ou subtraídos, dependendo de cada caso, múltiplos de 360^o de modo a que essa diferença de ângulos de rotação dos triângulos vizinhos fique abaixo dos 180^o.

Baxter et al [25] sugere ainda uma implementação para esta técnica com base numa estrutura *Stack*, onde o objetivo consiste em começar por analisar os triângulos das fronteiras dos polígonos, para que estes depois possam propagar as rotações corretas para o interior. Inicialmente é feito um *Push* de todos os triângulos da fronteira para a *Stack*, sendo estes marcados como visitados. Cada vez que é feito o *Pop* de um triângulo, vão ser calculados quais os seus vizinhos. Os que estejam marcados como visitados são ignorados, para os restantes vai ser verificado se existem inconsistências entre estes e o triângulo que inicialmente foi feito o *Pop*. Estes triângulos são marcados como visitados e é efetuado um *Push* de volta para a *Stack*, terminando o algoritmo quando a *Stack* estiver vazia.

Depois da aplicação do algoritmo anterior as rotações vão ser consistentes sempre que tal for possível, no entanto existe outra questão a ser abordada e que corresponde à segunda fase desta técnica: Minimização das rotações. Existe a possibilidade de as rotações resultantes do algoritmo anterior não serem mínimas, uma vez que o resultado depende do primeiro triângulo a ser analisado. A Figura 2.20 representa um exemplo onde as rotações são consistentes mas que ao mesmo tempo não são mínimas, devido a uma escolha errada do primeiro triângulo a ser analisado.

Figura 2.20: Exemplo com excesso de rotação, Fonte: [25]

A solução proposta para resolver a rotação excessiva consiste em fazer uma média dos ângulos de rotação atuais e somar ou subtrair múltiplos de 360 de modo a aproximar essa média o mais possível de 0. Este passo garante que qualquer excesso de rotação é eliminado. A Figura 2.21 exemplifica o *Morphing* de um gato, onde estão demonstradas as diferenças obtidas nas transformações com o método original e depois de aplicada a Consistência de Rotações descrita nesta subsecção. A primeira linha corresponde aos resultados do método original e podemos notar que existem inconsistências com triângulos vizinhos a fazer rotações em sentidos opostos, o que provocou uma transformação com deformação na cauda do gato. Na segunda linha estão representados os resultados da mesma transformação mas com aplicação da consistência de rotações, onde é possível verificar uma melhoria dos resultados, com a transformação a decorrer de forma natural e sem inconsistências entre triângulos vizinhos.

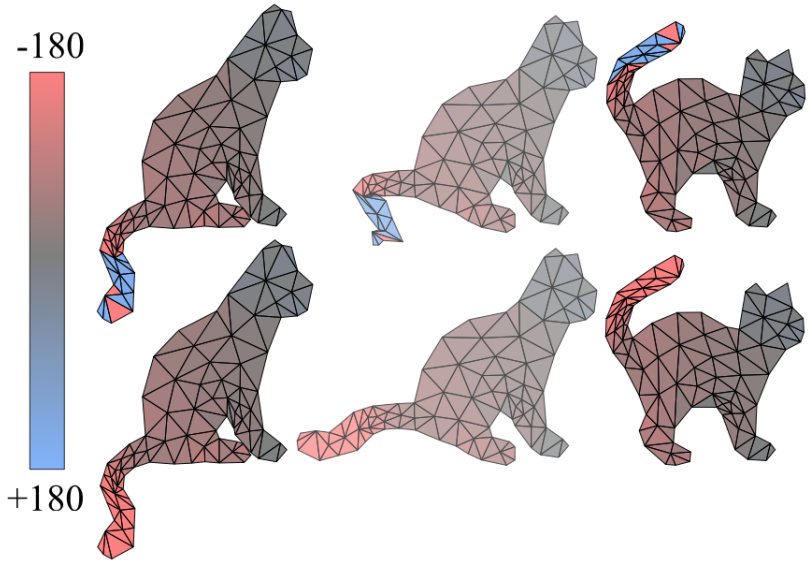


Figura 2.21: Diferenças depois da implementação, Fonte: [25]

2.4 SÍNTESE DO CAPÍTULO

Este capítulo teve como objetivo fazer uma introdução dos conceitos principais relacionados com a Triangulação e o *Morphing*.
O capítulo começa com uma introdução sobre Triangulação e foram abordadas duas técnicas: a Triangulação de Delaunay, técnica mais conhecida, e a Triangulação Compatível, técnica sobre a qual existiu maior detalhe. A Triangulação Compatível permite triangular dois polígonos com correspondências entre si, garantindo que no resultado final ambos os polígonos contém o mesmo número de triângulos, para que depois possa ser aplicado o *Morphing* sobre estes. O *Morphing* foi o outro foco deste capítulo, onde inicialmente foi feita uma introdução sobre a técnica, seguido de duas abordagens distintas para a resolução do *Vertex Path Problem*: Interpolação Linear e Interpolação Rígida, esta última com maior detalhe, uma vez que o seu objetivo é produzir uma transformação natural e que preserve as propriedades físicas do objeto.

capítulo 3

Técnicas de Morphing com Triangulação Compatível

3.1 introdução

Neste capítulo serão abordadas as implementações dos algoritmos escolhidos para estimar a transformação de um objeto móvel ao longo do tempo. O principal problema consiste em determinar a transformação a partir de um polígono origem e um polígono destino.

A Secção 3.2 descreve os métodos necessários para a geração de Triangulações Compatíveis para os polígonos origem e destino.

A Secção 3.3 apresenta os métodos implementados para realizar o *Morphing* de polígonos.

A Secção 3.4 descreve o desenvolvimento da interface gráfica.

A Secção 3.5 apresenta uma breve síntese sobre o trabalho desenvolvido ao longo deste capítulo.

3.2 TRIANGULAÇÃO COMPATÍVEL

A implementação para geração de triangulações compatíveis entre dois polígonos teve em conta o algoritmo proposto por *Surazhsky et al* [15], descrito na Secção 2.2.3, uma vez que este gera os resultados desejados utilizando Vértices de Steiner apenas quando necessário, o que resulta num melhor desempenho computacional. A solução apresentada admite que já existe uma correspondência de vértices entre os polígonos origem e destino, conhecido como *Vertex Correspondence Problem*, sendo esta definida pelos índices numéricos dos vértices. Este algoritmo foi implementado em *Matlab*, no ficheiro *compatibleTriangulation.m*, tendo o seu processo sido dividido em vários passos que serão descritos com detalhe nas subsecções seguintes.

3.2.1 OBTENÇÃO DO PAR DE VÉRTICES MAIS PRÓXIMOS

O primeiro passo consiste em fazer calculo das distâncias entre todos os vértices não vizinhos dos polígonos em análise. O objetivo é encontrar qual o par de vértices não vizinhos que estão mais próximos entre si (menor distância de ligação). Esta operação é realizada na função *calculateClosestVertex*, do ficheiro *compatibleTriangulation.m*, sendo que o seu conceito geral consiste em analisar os dois polígonos (origem e destino) e, para cada par de vértices não vizinhos, calcular as distâncias entre eles através da fórmula da distância Euclidiana.

Os dados obtidos são de seguida armazenados numa matriz que vai conter os resultados de todos os vértices analisados, onde cada linha tem, sob esta ordem, os seguintes dados:

- Identificador do polígono:
  - Caso seja 0, o polígono é o origem;
  - Caso seja 1, o polígono é o destino.
- Identificador numérico do primeiro vértice;
- Identificador numérico do segundo vértice;
- Distância entre os vértices.

No entanto existem casos onde é necessário ignorar o par de vértices em análise, nomeadamente:

- Ligação entre estes não é válida, de acordo com as regras de ligações inválidas já referidas na Secção 2.2.3, sendo para isso executado um algoritmo que tem como objetivo testar a validade da ligação (algoritmo na Secção 3.2.2);
- Já se encontram ligados através de um Vértice de Steiner;

- Estes pertencem à lista de vértices ignorados no ficheiro *ignoreList.txt*.

Caso alguma das condições descritas acima se verifique, então o par de vértices é ignorado, passando imediatamente para o próximo par de vértices a ser analisado. Depois de obtida a matriz final, com todas as combinações válidas de vértices e as respectivas distâncias entre si, esta vai ser ordenada por ordem crescente da distância calculada.

Depois de ordenada, a primeira linha da matriz contém o par de vértices mais próximos e é necessário verificar se a mesma ligação no outro polígono é válida, sendo para isso utilizado o mesmo algoritmo de validação da ligação referido anteriormente. Quando essa ligação não é válida, situação já exemplificada anteriormente nas Figuras 2.10 e 2.11, é necessário introduzir um Vértice de Steiner (algoritmo na Secção 3.2.3), de modo a contornar a situação e formar uma ligação válida.

É importante referir que o algoritmo de cálculo dos Vértices de Steiner pode não conseguir calcular um vértice válido. Quando isso acontece, o par de vértices que se está a tentar ligar é ignorado e adicionado ao ficheiro *ignoreList.txt*, que contém uma lista todos os pares de vértices onde não foi possível calcular Vértices de Steiner válidos. Deste modo, nas chamadas recursivas seguintes da Triangulação Compatível, o algoritmo não vai voltar a tentar efetuar uma ligação sobre este par de vértices. É então utilizado o par de vértices seguinte da matriz final obtida anteriormente, sendo o processo repetido até a ligação no polígono correspondente seja possível (com ou sem recurso a Vértices de Steiner).

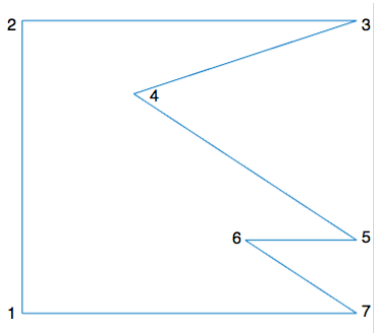
3.2.2 VALIDAÇÃO DA LIGAÇÃO

O algoritmo para a validação da ligação de vértices está implementado na função *checkPointConnection*, do ficheiro *compatibleTriangulation.m*, que recebe como argumentos:

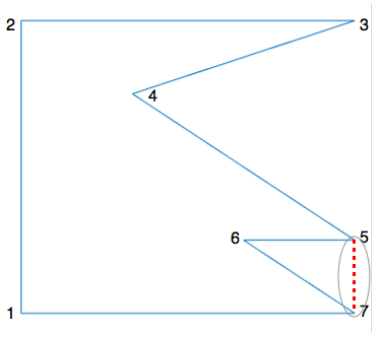
- *point1*: coordenadas do primeiro vértice;
- *point2*: coordenadas do segundo vértice;
- *polygon*: coordenadas do polígono onde está a ser testado se a ligação é válida.

A conceito geral do algoritmo implementado consiste em obter um conjunto de pontos intermédios entre o par de vértices sob o qual se quer testar a ligação. Para cada um dos pontos intermédios calculados é testado se estes se encontram dentro, fora ou na fronteira do polígono, através da função *inpolygon*, do *Matlab*. Caso seja detetado que algum destes se encontra fora ou na fronteira do polígono, a ligação é considerada inválida.

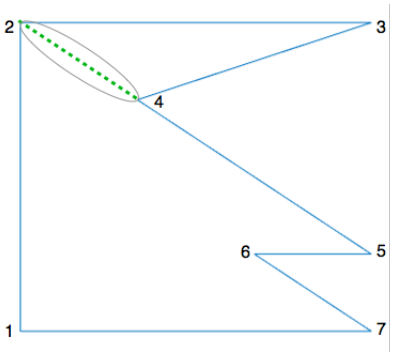
Na Figura 3.1 podemos ver, num exemplo já referido anteriormente, a abordagem do algoritmo implementado. Inicialmente é detetado que o par de vértices mais próximos entre si são o par 5-7, então vão ser calculados os pontos intermédios, representados dentro da elipse na Figura 3.1b. Neste conjunto de pontos intermédios, existem pontos fora do polígono, pelo que a ligação é considerada inválida, sendo assim ignorada. De seguida é considerado o par 2-4 como aqueles que se encontram mais próximos entre si, Figura 3.1c. O processo anterior é repetido, mas, desta vez, todos os pontos intermédios encontram-se dentro do polígono, resultando assim numa ligação que é considerada válida.



(a) Polígono Original



(b) Ligação Inválida



(c) Ligação Válida

Figura 3.1: Validação da ligação de vértices

3.2.3 INTRODUÇÃO DE VÉRTICES DE STEINER

O algoritmo de procura de Vértices de Steiner consiste numa procura circular de pontos e foi implementado na função *findSteiner*, do ficheiro *compatibleTriangulation*, que recebe como argumentos:

- *point1*: coordenadas do primeiro vértice;
- *point2*: coordenadas do segundo vértice;
- *polygon*: coordenadas do polígono onde se quer introduzir o Vértice de Steiner;

- *iteration*: número de iterações, que será um fator que vai influenciar o raio da pesquisa circular;
- *flag*: identificador que indica se o polígono em questão é o origem ou destino;
- *deepSearch*: variável que indica se é para utilizar a pesquisa leve ou profunda (detalhadas em seguida).

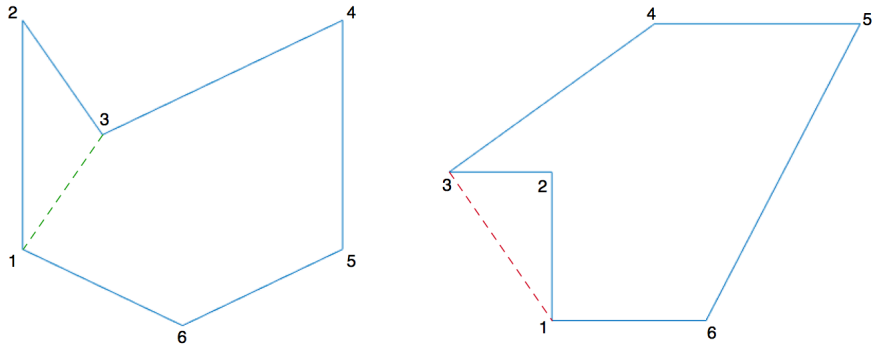
O processo inicia-se com o calculo do ponto médio entre o par de vértices cuja ligação é inválida, sendo este ponto utilizado como centro da circunferência. Em relação ao raio inicial da circunferência, este é calculado mediante a distância entre o par de vértices, desta forma é possível ter um valor dinâmico que se ajusta conforme a distância a que estão os vértices.

Depois de calculadas as componentes do circunferência (centro e raio inicial) vão ser calculados, através da Equação Paramétrica, pontos em vários ângulos à volta da circunferência. Para esta pesquisa circular de pontos foram implementados dois tipos de pesquisa: leve e profunda. Cada uma das pesquisas referidas atua, no máximo, durante vinte iterações (para evitar possíveis execuções infinitas), iterações estas que influenciam o raio da procura circular: quanto maior o número de iterações, maior o raio da circunferência. A pesquisa leve é a primeira a ser utilizada, começando inicialmente a procurar pontos a cada 45º. No caso de não encontrar nenhum válido irá voltar a procurar pontos a cada 35º e, no pior caso, irá procurar pontos a cada 25º. No caso de, ao fim das vinte iterações, não ter sido encontrado nenhum Vértice de Steiner válido com a pesquisa leve, então é utilizada a pesquisa profunda, que começa por procurar pontos válidos a cada 10º e, no pior caso, irá procurar a cada 1º. Quando a pesquisa durante o pior caso (a cada 25º na pesquisa leve e a cada 1º na pesquisa profunda) termina uma volta inteira na circunferência sem encontrar um Vértice de Steiner válido, então vai ser efetuada uma nova iteração da pesquisa, ou seja, esta recomeça com o ângulo inicial (45º na pesquisa leve e 10º na pesquisa profunda) mas o raio da circunferência vai ser maior, situação representada na Figura 3.3.

Para cada ponto obtido anteriormente, é necessário testar se existe uma ligação válida com o par de vértices, usando o algoritmo de validação já referido anteriormente. No caso de inválida, a pesquisa irá continua a procurar. No caso da ligação ser válida, o algoritmo de pesquisa termina, sendo necessário calcular as coordenadas do Vértice de Steiner no outro polígono, que será o ponto médio entre o par de vértices.

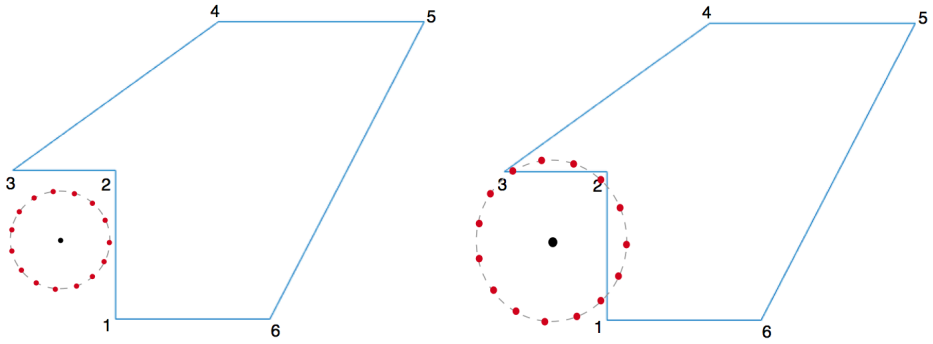
Por último, depois de obtidas as coordenadas dos Vértices de Steiner em ambos os polígonos, estas são guardadas no ficheiro *steiner.txt*, que contém as informações sobre todos os Vértices de Steiner calculados. As informações guardadas são: identificadores numéricos do par de vértices ligados pelo Vértice de Steiner, coordenadas do Vértice de Steiner no polígono origem, coordenadas do Vértice de Steiner no polígono destino e identificador numérico do Vértice

de Steiner. Para os identificadores numéricos dos Vértices de Steiner são utilizados índices a começar em N+1, onde N = número de vértices do polígono.

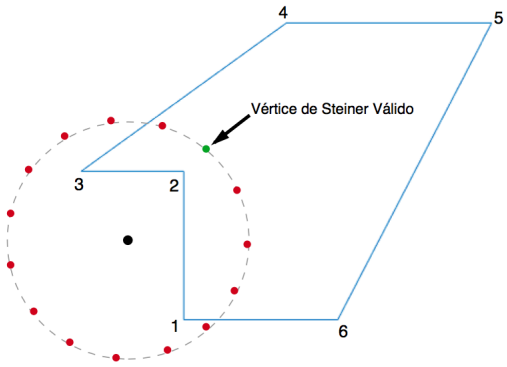


(a) Polígono Origem (b) Polígono Destino

Figura 3.2: Procura de Vértice de Steiner: Par de vértices mais próximos

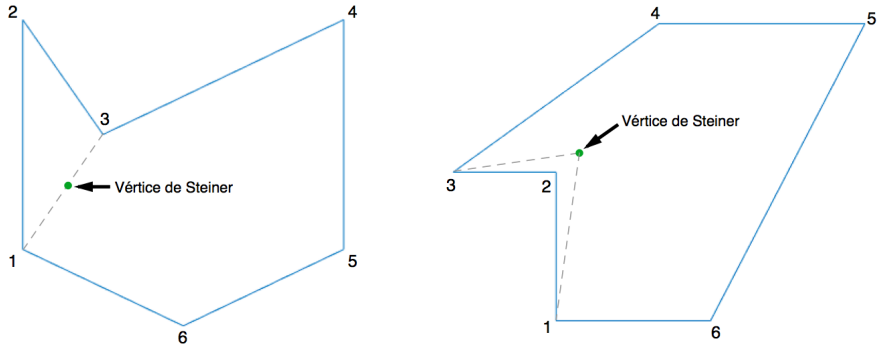


(a) Polígono Destino: Iteração 1 (b) Polígono Destino: Iteração 2



(c) Polígono Destino: Iteração 3

Figura 3.3: Procura de Vértice de Steiner: Iterações na pesquisa



(a) Polígono Origem: Resultado (b) Polígono Destino: Resultado

Figura 3.4: Procura de Vértice de Steiner: Representação dos resultados obtidos

As Figuras 3.2, 3.3 e 3.4 representam o conceito geral do algoritmo implementado. Inicialmente é detetado, aplicando o algoritmo de procura de vértices mais próximos entre si, que o par de vértices mais próximos são o 1-3 do polígono origem, representado na Figura 3.2a. No entanto a ligação correspondente não é válida, como é possível verificar na Figura 3.2b, sendo por isso necessário encontrar um Vértice de Steiner para tornar essa ligação válida. Na Figura 3.3a podemos ver a primeira iteração da pesquisa, onde é feita uma pesquisa circular de pontos, pontos estes representados na fronteira da circunferência. Uma vez que nenhum destes é válido, é necessário fazer uma nova iteração, onde o raio de pesquisa irá ser aumentado. A Figura 3.3b representa a segunda iteração da pesquisa, onde o raio de pesquisa foi aumentado, verificando-se neste caso que já existem pontos que se encontram dentro do polígono. Apesar disso, nenhum deles cria uma ligação válida com o par de vértices 1-3, pelo que é necessário nova iteração, com consequente aumento do raio de pesquisa. Essa nova iteração está representada na Figura 3.3c, onde se verifica que já existe um ponto que cria uma ligação válida (representado com cor verde) entre o par de vértices 1-3, ou seja, está encontrado o Vértice de Steiner a ser utilizado, terminando o algoritmo de pesquisa assim que este ponto é encontrado. Na Figura 3.4b podemos ver o resultado depois de efetuada a ligação com o Vértice de Steiner encontrado, assim como a ligação correspondente, representada na Figura 3.4a, onde o Vértice de Steiner, como já referido anteriormente, é o ponto médio entre o par de vértices, neste caso o par 1-3.

3.2.4 GERAÇÃO DOS SUB-POLÍGONOS

Depois de calculado o par de vértices não vizinhos mais próximos entre si (menor distância de ligação), é necessário gerar os dois sub-polígonos que essa ligação de vértices vai criar. Para essa geração foi desenvolvido um algoritmo, implementado na função *constructPolygon*, do ficheiro *compatibleTriangulation.m*, que recebe como argumentos:

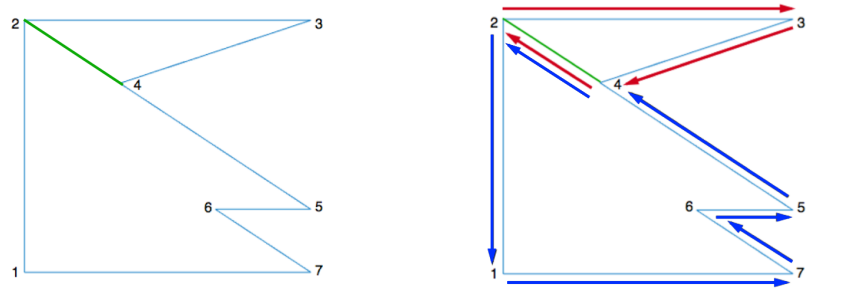
- *polygon*: Matriz de vértices do polígono que vai ser particionado;
- *point1*: Vértice inicial da ligação;
- *point2*: Vértice final da ligação;
- *steinerIndex*: Identificador do Vértice de Steiner adicionado, caso nenhum tenha sido adicionado esta variável tem o valor 0.

O fluxo do algoritmo consiste em duas fases semelhantes:

- Geração do caminho frontal
  - Esta fase inicia-se com a procura do vértice inicial da ligação na matriz de vértices do polígono. Depois de encontrado, são percorridos (por ordem crescente na matriz) e guardados todos os identificadores dos vértices intermédios até chegar ao vértice final de ligação. Em seguida, todos os identificadores dos vértices percorridos são escritos no ficheiro *triangles.txt* e que contém uma lista de todos os sub-polígonos gerados até ao momento.
- Geração do caminho inverso
  - Esta fase é semelhante à anterior, com a diferença que quando se encontra o vértice inicial da ligação na matriz de vértices se percorre de forma decrescente a matriz com os vértices intermédios até chegar ao vértice final da ligação. No final também é escrito no ficheiro *triangles.txt* o polígono gerado.

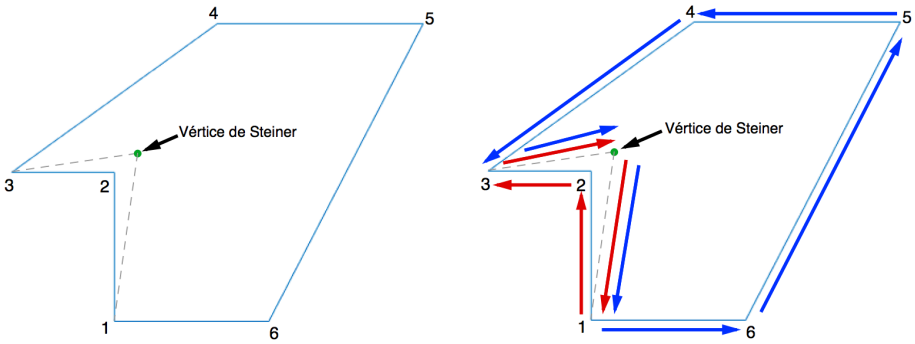
O algoritmo age de forma ligeiramente diferente mediante o valor do identificador do Vértice de Steiner recebido como argumento. Caso este seja 0, significa que não foi introduzido nenhum Vértice de Steiner na ligação e o fluxo da partição é o descrito acima. Caso seja diferente de 0, então foi introduzido um Vértice de Steiner e esta variável contém o valor do seu identificador numérico. A diferença relativamente ao fluxo descrito anteriormente é que também é necessário guardar o identificador do Vértice de Steiner no ficheiro *triangles.txt*, uma vez que este vértice agora também faz parte dos dois sub-polígonos gerados, sendo este guardado logo na primeira posição.

O processo sem adição de Vértices de Steiner é ilustrado na Figura 3.5, onde a geração do caminho frontal está definida pelas setas a vermelho (vértices 2-3-4) e a geração do caminho inverso definida pelas setas a azul (vértices 2-1-7-6-5-4). O processo com Vértice de Steiner pode observar-se na Figura 3.6, onde o caminho frontal está representado a vermelho (vértices STEINER-1-2-3) e o caminho inverso representado a preto (vértices STEINER-1-6-5-4-3).



(a) Polígono para particionar (b) Conceito geral da partição

Figura 3.5: Geração dos sub-polígonos



(a) Polígono para particionar (b) Conceito geral da partição

Figura 3.6: Geração dos sub-polígonos com Vértices de Steiner

Depois de efetuada a geração dos sub-polígonos é necessário verificar se existem polígonos, com mais de três vértices, armazenados no ficheiro *triangles.txt*. Esta verificação é feita na função *checkSubPolygons*, onde vai ser executada uma chamada recursiva para triangular qualquer polígono com mais de três vértices. O algoritmo de Triangulação Compatível termina quando todos os polígonos armazenados no ficheiro *triangles.txt* tenham apenas três vértices, ou seja, já são todos triângulos.

3.2.5 SUAVIZAÇÃO DA MALHA DE TRIÂNGULOS

Foi desenvolvido um algoritmo de Suavização para melhorar a malha de triângulos inicialmente calculada, sendo que o seu funcionamento consiste numa procura circular de pontos (similar à procura de Vértices de Steiner referida na Secção 3.2.3) que podem ser alternativas aos Vértices de Steiner atuais. O fator de melhoria pode ser escolhido entre a maximização dos ângulos mínimos ou a maximização das áreas mínimas da malha de triângulos, ou seja, sempre que se consegue aumentar estes valores mínimos, então é considerado que este novo ponto melhora a malha. Uma malha com triângulos longos e finos tem pouca qualidade, estas características resultam de triângulos com ângulos mínimos muito pequenos, pelo que é o fator de maximização mais importante. A maximização das áreas

mínimas foi implementada para que fosse possível verificar as diferenças em comparação com o fator anterior.

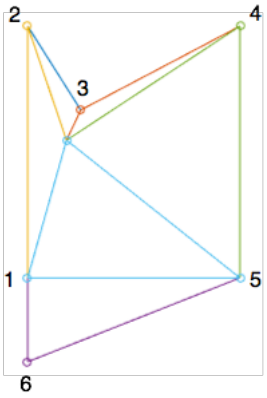
Este algoritmo está disponível no ficheiro *smoothTriangulation.m* e utiliza como argumento de entrada:

- *flagSmooth*: fator de maximização a ser utilizado.
  - flagSmooth=0: o objetivo será maximizar o ângulo mínimo;
  - flagSmooth=1: o objetivo será maximizar a área mínima.

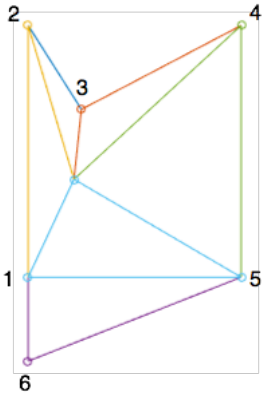
O algoritmo começa por fazer uma leitura de todos os Vértices de Steiner existentes, onde, para cada vértice, vai ser obtida a lista de triângulos aos quais pertence, de modo a que possa ser calculado o ângulo mínimo ou área mínima da malha atual, mediante o fator de minimização escolhido.

De seguida, faz-se uma pesquisa circular, usando a Equação Paramétrica, com centro no Vértice de Steiner e raio definido pela variável *searchRadius* (que é uma variável que vai decrementando automaticamente), para obter o conjunto de pontos sob os quais vão ser testados se melhoram o fator de maximização atual. Para cada ponto obtido é verificado se aumentam o valor mínimo atual. Em caso afirmativo são armazenados numa lista final, onde também é incluído o seu valor mínimo para que, no final, esta lista possa ser ordenada de forma decrescente pelo valores mínimos, de modo a que os pontos com maior maximização sejam os primeiros da lista.

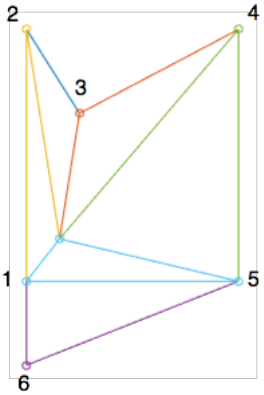
O último passo consiste em percorrer a lista calculada até encontrar o primeiro ponto que forme uma malha de triângulos válida (uma vez que a malha está ordenada pelos pontos com maior fator de maximização), sendo esta verificação feita através do algoritmo descrito na Secção 3.2.6. Este processo é repetido até não existirem pontos que maximizem o fator escolhido mais que o atual, o que significa que foi encontrado o melhor ponto para esse Vértice de Steiner. Nos casos em que existe mais do que um Vértice de Steiner é necessário ter em conta que a melhoria de um vértice pode ter alterações nas melhorias já calculadas para outros vértices. Para resolver essas situações foi utilizada uma variável para indicar sempre que ocorre uma mudança global da malha, ou seja, enquanto houver alterações na malha, independentemente do vértice, o algoritmo volta a ser executado até que não exista nenhuma alteração em nenhum Vértice de Steiner. Na Figura 3.7a podemos ver um exemplo de um polígono com uma malha mal formada e nas Figuras 3.7b e 3.7c os resultados depois da aplicação do algoritmo descrito com o objetivo de maximizar, respetivamente, o ângulo mínimo e a área mínima.



(a) Polígono antes da melhoria



(b) Maximização do ângulo mínimo



(c) Maximização da área mínima

Figura 3.7: Exemplo de melhoria da malha de triângulos

Um resumo do fluxo do algoritmo pode ser visto no pseudocódigo seguinte:

```
while globalChanges==1 do
    globalChanges=0;
    for each Steiner Point do
        while vertexChanges==1 do
            vertexChanges=0;
            Get list of triangles with the Steiner Point;
            Calculate list of circular points;
            Store points that maximize the fator in a final list;
            Descending order the final list by minimum fator value;
            for each point in final list do
                if point generates valid mesh then
                    Substitute in the file steiner.txt;
                    vertexChanges=1;
                    globalChanges=1;
                    break;
                end
            end
        end
    end
end
```

Algoritmo 1: Pseudocódigo com resumo do fluxo da Suavização

3.2.6 DETEÇÃO DE INTERSEÇÕES

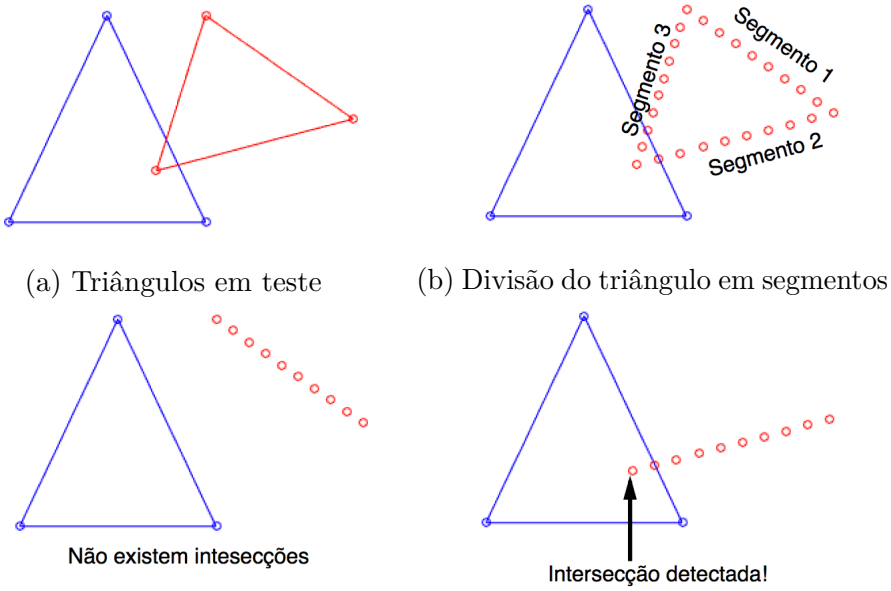
O algoritmo desenvolvido para a deteção de interseções numa malha de triângulos está implementado no ficheiro *detectIntersections.m*, que recebe como argumento uma variável com o nome *triangulatedPolygon*. Esta variável é uma matriz que contém as coordenadas dos vértices de todos os triângulos da malha, ou seja, tendo em conta que cada triângulo ocupa três linhas da matriz, então esta tem dimensão Nx2, onde N = número de triângulos*3.

O conceito geral deste algoritmo consiste em percorrer a matriz *triangulatedPolygon* e testar, para todas as combinações existentes entre os vários triângulos, se existem interseções. As combinações entre os triângulos são obtidas através de dois ciclos, onde cada iteração percorre as três posições da matriz de onde vão ser extraídas as coordenadas dos triângulos aos quais vão ser testadas as interseções.

No teste das interseções entre dois triângulos, um destes vai ser dividido em três segmentos, onde cada segmento vai ser dividido num conjunto de pontos, que vão ser analisados para saber se algum deles se encontra dentro da área do outro triângulo, o que significa que existe

uma interseção. Através da função *inpolygon*, do *Matlab*, é verificado se algum dos pontos calculados se encontra dentro do triângulo.

Na Figura 3.8 está representado um exemplo de interseção entre dois triângulos, onde o triângulo vermelho foi dividido em três segmentos, segmentos que por sua vez são subdivididos em vários pontos, sendo necessário verificar se algum destes pontos se encontra dentro do triângulo azul. Esta situação acaba por acontecer, neste exemplo, para o segmento dois, não sendo necessário continuar a testar para o segmento seguinte e terminando com a variável *hasIntersection* com o valor '1', que sinaliza a existência de interseções.



(a) Triângulos em teste (b) Divisão do triângulo em segmentos

(c) Testes para o primeiro segmento (d) Testes para o segundo segmento

Figura 3.8: Exemplo de deteção de interseções

Um resumo do fluxo do algoritmo é apresentado no pseudocódigo seguinte:

```
while i<=size(triangulatedPolygon)-3 AND hasIntersection==0 do
    j=i+3;
    while j<=size(triangulatedPolygon) AND hasIntersection==0 do
        Read the two triangles;
        for Each segment do
            Read segment vertexes;
            Generate set of points connecting the vertexes;
            if Point inside triangle area then
                hasIntersections=1;
                break;
            end
        end
        j=j+3;
    end
    i=i+3;
end
```

Algoritmo 2: Pseudocódigo com resumo do algoritmo de deteção de interseções

3.2.7 FICHEIROS UTILIZADOS

No algoritmo desenvolvido para a Triangulação Compatível são utilizados vários ficheiros, que são armazenados na pasta *Files*, sendo aqui feito um breve resumo dos objetivos de cada um deles:

- *source.txt*
  - Este ficheiro contém as coordenadas do polígono origem, onde cada linha corresponde a um vértice.
- *target.txt*
  - Este ficheiro contém as coordenadas do polígono destino, onde cada linha corresponde a um vértice.
- *triangles.txt*
  - Neste ficheiro são armazenados os resultados do algoritmo da Triangulação Compatível, onde cada linha do ficheiro contém os identificadores numéricos de cada triângulo.
- *steiner.txt*
  - Este ficheiro é utilizado para armazenar todas as informações necessárias relativas aos Vértices de Steiner.

- *auxiliar.txt*
  - Ficheiro utilizado como auxílio quando é necessário apagar alguma linha do ficheiro *triangles.txt*.
- *ignoreList.txt*
  - Este ficheiro é utilizado para armazenar os pares de vértices para os quais não foi possível estabelecer uma ligação válida, de modo a evitar que execuções recursivas da Triangulação Compatível voltem a tentar efetuar uma ligação que anteriormente já tinha sido falhada.

3.3 MORPHING

Para a implementação da técnica de *Morphing* foram utilizados os algoritmos de Interpolação Rígida propostos por *Alexa et al* [6] e *Baxter et al* [25], devido ao facto destes terem como principal objetivo a preservação das propriedades físicas do objeto, evitando deformações, que é um dos pormenores mais importante numa boa técnica de *Morphing*. Tal como na solução da Triangulação Compatível, a solução de *Morphing* foi implementada em *Matlab*, no ficheiro *interpolation.m*, admitindo que também já existe uma correspondência de vértices entre os polígonos, conhecido como *Vertex Correspondence Problem*, correspondência esta atribuída pelos identificadores numéricos dos vértices.

Na solução desenvolvida estão disponíveis três tipos de interpolação:

- uma Interpolação Linear, detalhada na Secção 3.3.1;
- duas de Interpolação Rígida, detalhadas nas Secções 3.3.2 e 3.3.3.

Nas outras subsecções são descritas resoluções para os problemas encontrados durante as implementações práticas das técnicas de interpolação referidas.

3.3.1 INTERPOLAÇÃO LINEAR

O conceito geral do algoritmo implementado para a Interpolação Linear consiste em fazer transformações por iteração, onde em cada iteração é efetuada uma transformação, para um dado instante entre 0 e 1, de um triângulo origem para um triângulo destino, sendo que o instante 0 corresponde ao triângulo origem e 1 ao triângulo destino.

Este algoritmo está implementado na função *linearInterpolation*, do ficheiro *interpolation.m*, que recebe como argumento o instante da transformação. O algoritmo começa com

a leitura do ficheiro *triangles.txt*, onde estão armazenados os identificadores numéricos que formam cada triângulo. Uma vez que este ficheiro contém apenas os identificadores dos triângulos, então é necessário convertê-los para os respetivos triângulos origem e destino. Para isso foi desenvolvida uma função denominada *getTrianglesFromIndex* que recebe uma lista dos identificadores numéricos e faz a conversão para as matrizes de coordenadas, sob as quais vai ser feita a transformação.

Para cada par de triângulos origem e destino obtidos, é necessário fazer a respectiva transformação, sendo esta efetuada através da função *getTriangleOnInstantLinear*, que utiliza como argumentos as coordenadas do triângulo origem, destino e o instante da transformação. Dentro desta função vai ser calculada a matriz afina que vai fazer a transformação do triângulo origem para o triângulo destino, sendo a obtenção da matriz afina descrita com detalhe na Secção 3.3.4. Esta matriz tem dimensão 3x3, uma vez que também está incluída a componente de Translação, tal como apresentado na Equação 2.2. De forma a que seja possível obter as transformações intermédias, é necessário variar esta matriz no espaço temporal, A(t), para possa ser feita a transformação num determinado instante entre [0 ; 1], representação esta visível na Equação 3.1.

$$
\begin{aligned}
A &= \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \\
A(t) &= (1-t) * I + t * A \\
\Leftrightarrow A(t) &= (1-t) * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + t * \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

O passo final consiste em fazer as operações apresentadas na Equação 2.3, ou seja, alterar a estrutura da matriz de coordenadas do triângulo origem para 3x3, adicionando uma linha de 1's (porque a matriz afina já contem a componente de Translação), seguida de uma multiplicação da matriz afina, no instante t, por esta nova matriz de coordenadas. Por exemplo, para obter as coordenadas intermédias de uma transformação, no instante t, de um triângulo origem S num triângulo destino T:

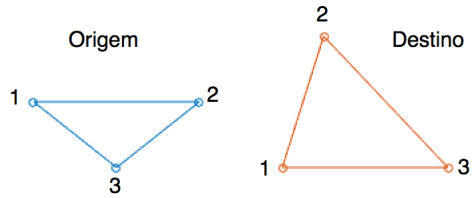
$$
\begin{aligned}
T(t) &= A(t) * S \\
\Leftrightarrow T(t) &= \begin{bmatrix} at_{11} & at_{12} & tt_x \\ at_{21} & at_{22} & tt_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} s_{x1} & s_{x2} & s_{x3} \\ s_{y1} & s_{y2} & s_{y3} \\ 1 & 1 & 1 \end{bmatrix}
\end{aligned}
\tag{3.2}
$$

Uma síntese do algoritmo é apresentado abaixo, em pseudocódigo, seguido de um exemplo de *Morphing*, utilizando o algoritmo de Interpolação Linear descrito, na Figura 3.9. A azul

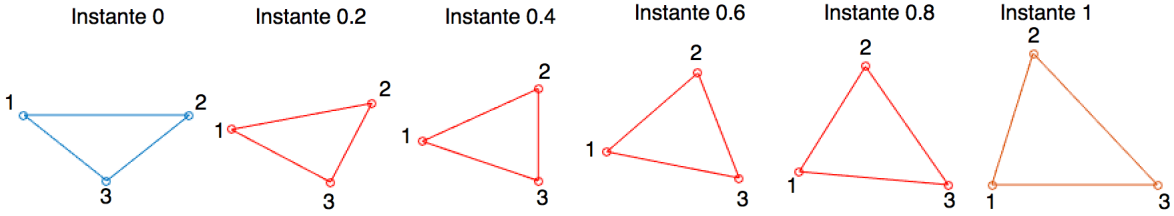
está representado o triângulo origem, a laranja o triângulo destino e a vermelho as formas intermédias.

for *each line in triangles.txt* **do**
    [source,target] = getTrianglesFromIndex(index);
    A = calculateAffineMatrix(source,target);
    A(t) = (1-t)*I+t*A;
    source = source + row with 1's;
    coordinates=A(t)*source;
end

Algoritmo 3: Resumo do algoritmo de Interpolação Linear



(a) Polígonos Origem e Destino



(b) Transformações intermédias

Figura 3.9: Resultados da Interpolação Linear

3.3.2 INTERPOLAÇÃO RÍGIDA MÉDIA

Nesta subsecção é descrita a implementação da Interpolação Rígida Média, que é uma técnica onde para cada par de triângulos origem-destino são geradas as suas transformações ótimas, já discutidas na Secção 2.3.3, sendo as posições finais dos vértices partilhados calculada através da média de todas as suas coordenadas. Esta implementação corresponde à função *optimalMeanInterpolation*, do ficheiro *interpolation.m*, que recebe como argumentos:

- instant: instante da transformação;
- angleConsistency: variável referente à utilização de rotações consistentes. Caso o valor desta seja '0' o algoritmo segue o fluxo normal, caso seja '1' vai seguir um fluxo adaptado de modo a obter rotações consistentes sempre que possível, de acordo com o algoritmo abordado na Secção 2.3.5.

O fluxo normal do algoritmo inicia com uma leitura do ficheiro *triangles.txt* e com as conversões para os respetivos triângulos. Uma vez que a translação é efetuada em separado, os triângulos são inicialmente centrados na origem, como demonstrado na Figura 3.10. Neste exemplo, o triângulo inicial está representado com a cor azul e o mesmo triângulo depois de efetuada a translação para a origem está representado a vermelho.

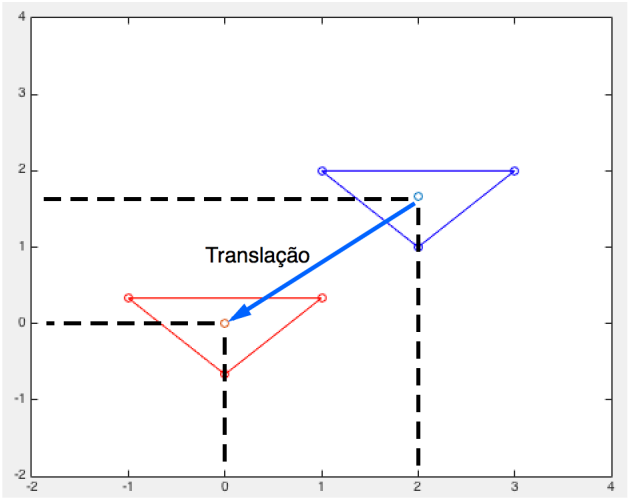


Figura 3.10: Exemplo de translação para a origem

Em seguida, é necessário obter a matriz que vai gerar a transformação ótima. O processo de obtenção desta matriz está implementado na função *calculateOptimalTransformation*, que recebe como argumentos:

- source: coordenadas do triângulo origem;
- target: coordenadas do triângulo destino;
- instant: instante da transformação.

Através das matrizes de coordenadas de ambos os triângulos é calculada matriz afina que efetua a transformação de um no outro, através do processo descrito na Secção 3.3.4. A matriz obtida tem dimensão 3x3, uma vez que já tem a componente de Translação incluída, componente esta que é ignorada com a eliminação da linha 3 e coluna 3 da matriz de transformação, ficando com dimensão 2x2, o formato desejado na implementação desta técnica.

A decomposição da matriz afina, através do *Singular Value Decomposition*, é efetuada com recurso à função *SVD* [32] do *Matlab*. Esta vai gerar as três matrizes, já apresentadas anteriormente na Equação 2.6, através das quais é possível obter as componentes R_γ e S, utilizadas na geração da matriz afina ótima, A_γ na Equação 2.7. Para ser possível obter a transformação ótima para qualquer instante, é necessário fazer a variação no espaço temporal, t, desta matriz. No entanto, esta variação não pode ser feita diretamente sobre a matriz, uma

vez que pode gerar resultados errados em certas situações, sendo esta componente descrita com detalhe na Secção 3.3.6.

Depois de obtida a matriz afina ótima para o instante pretendido, é efetuada a multiplicação desta com as coordenadas do triângulo origem, representadas numa matriz com dimensão 2x3, onde o resultado são as novas coordenadas do triângulo para o instante t. Estas encontram-se centradas na origem, sendo ainda necessário fazer a Interpolação da Translação, de modo a colocar o novo triângulo na posição correta, sendo esta componente abordada na Secção 3.3.5.

Como estamos a lidar com transformações ótimas surge um problema com a posição dos vértices partilhados por triângulos. Este problema deve-se ao facto dos vértices partilhados não terem uma coordenada fixa, dado que a transformação ótima de um triângulo pode colocar um vértice numa posição e a transformação ótima de outro triângulo pode colocar esse mesmo vértice noutra posição, ou seja, aquilo que acontece é que, ao longo transformação, os triângulos vão separar-se, voltando a juntar-se no final quando é gerado o polígono destino. É possível verificar este problema na Figura 3.11.

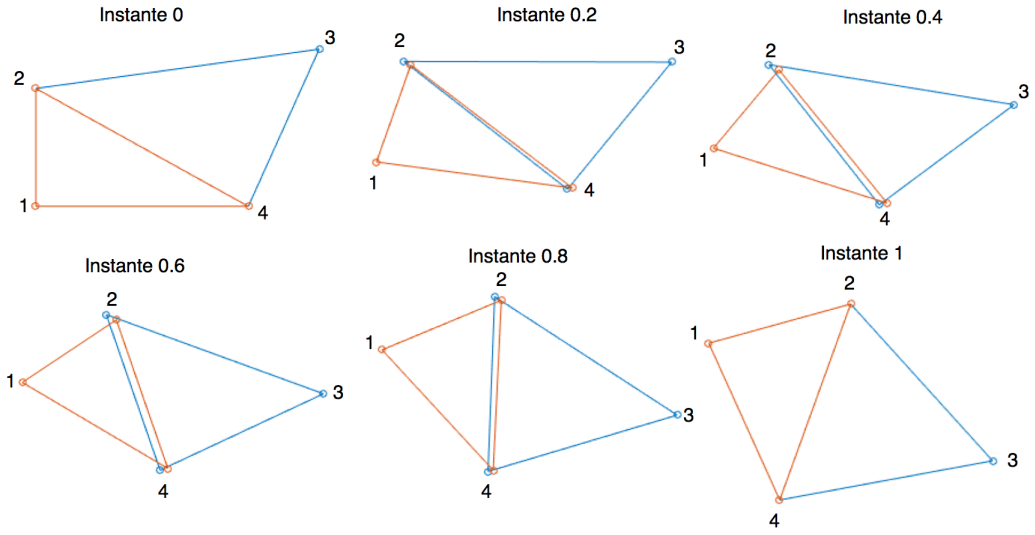


Figura 3.11: Separação dos triângulos com vértices partilhados

Para resolver este problema foi implementado um algoritmo para gerar uma posição única para cada vértice, com base na média das coordenadas que cada vértice partilhado tem, sendo este dividido em duas fases. A primeira fase está implementada na função *analyseVertexInformation* onde, para cada triângulo calculado, são analisadas as posições dos seus vértices e que vão ser escritas no ficheiro *vertexPositions.txt*, juntamente com o índice do vértice. O objetivo é que, no final, estejam as várias coordenadas que cada vértice teve, onde cada linha terá o formato [índiceVertice, coordenada1, coordenada2, ... , coordenadaN].
A segunda fase está implementada na função *calculateMeanCoordinates*, que consiste na leitura do ficheiro *vertexPositions.txt* e calcular a média das coordenadas para gerar a coordenada

final de cada vértice partilhado, excepto para os instantes 0 e 1, uma vez que estes instantes correspondem, respetivamente, aos polígonos origem e destino e, por isso, não existem vértices partilhados com coordenadas diferentes.

A seguir é apresentada uma síntese, em pseudocódigo, do algoritmo implementado, seguido da Figura 3.12, onde são apresentados obtidos por este algoritmo, para o mesmo exemplo utilizado anteriormente.

for *each line in triangles.txt* **do**
    [source,target] = getTrianglesFromIndex(index);
    source = translateToOrigin(source);
    target = translateToOrigin(target);
    optimalAffine = calculateOptimalTransformation(source,target,instant);
    coordinates=optimalAffine*source;
    Add new coordinates to the results matrix;
    analyseVertexInformation(coordinates);
end
results = calculateMeanCoordinates();

Algoritmo 4: Síntese do algoritmo de Interpolação Rígida Média

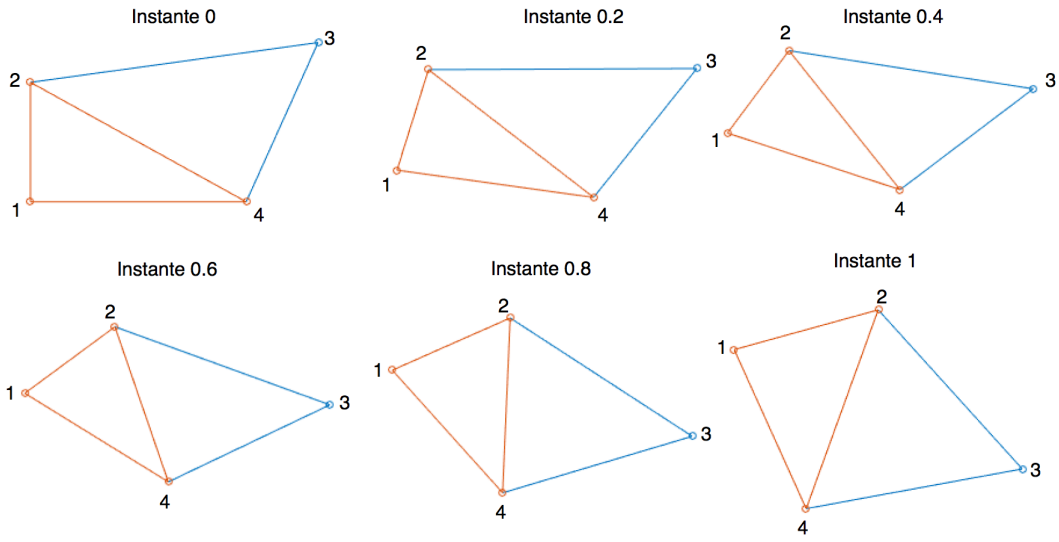


Figura 3.12: Solução com média das coordenadas

3.3.3 INTERPOLAÇÃO RÍGIDA ÓTIMA

Nesta subsecção é descrita a implementação do terceiro e último algoritmo de interpolação. O método tem semelhanças com o descrito na Secção 3.3.2, uma vez que são utilizadas as

mesmas transformações ótimas para cada par de triângulos origem-destino existentes, no entanto o cálculo das posições finais dos vértices partilhados é feita com recurso ao algoritmo já discutido na Secção 2.3.4.

A implementação deste algoritmo está na função *optimalInterpolation*, do ficheiro *interpolation.m*, que recebe dois argumentos:

- instant: instante da transformação;
- angleConsistency: variável referente à utilização de rotações consistentes. Caso o valor desta seja '0' o algoritmo segue o fluxo normal, caso seja '1' vai seguir um fluxo adaptado de modo a obter rotações consistentes sempre que possível, de acordo com o algoritmo abordado na Secção 2.3.5.

O fluxo normal do algoritmo começa com uma leitura do ficheiro *triangles.txt* e com as conversões para os respetivos triângulos. No entanto o objetivo de cada iteração é, para cada par de triângulos origem-destino, calcular as suas componentes individuais $\bar{P}$ e $A_\gamma(t)$ que, no final, vão gerar as matrizes globais utilizadas na resolução da equação normal, mais concretamente a matriz global $\bar{P}$, da Equação 2.15, e a matriz global $\bar{A}(t)$, da Equação 2.16. Estas matrizes globais correspondem, no código implementado, às variáveis:

- sparseGroup: matriz global $\bar{P}$;
- affineGroup: matriz global $\bar{A}(t)$.

Para o cálculo da matriz afina que gera a transformação ótima, A_γ, é utilizada a função *calculateOptimalTransformation*, já utilizada na Secção 3.3.2, sendo depois necessário obter a sua variação no espaço temporal, $A_\gamma(t)$, através do processo referido na Secção 3.3.6, onde o resultado é armazenado na variável *optimalAffine*. No cálculo da componente individual $\bar{P}$, é utilizada a função *calculateSparsePVector* que recebe como argumentos de entrada:

- sourceTriangle: matriz, com dimensões 3x2, que contém as coordenadas do triângulo origem;
- indexList: matriz, com dimensões 3x1, que contém os identificadores numéricos do vértices do triângulo origem.

Dentro desta função, inicialmente é calculada a matriz $\hat{P}$, já apresentada na Equação 2.10. Em termos de código *Matlab*, existe mais do que uma hipótese para realizar o cálculo desejado. Foi utilizado o formato apresentado na Equação 3.3, por sugestão do *Matlab*, que refere questões de rapidez e precisão na utilização deste formato, sendo o resultado desta operação armazenado na variável *pStar*.

$$pStar = (D * sourceTriangle) \backslash D \tag{3.3}$$

$$pStar = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \end{bmatrix}$$

De seguida é criada a matriz esparsa de dimensões 2xN, onde N corresponde ao número de vértices total (Normais + Steiner) do polígono. Inicialmente esta matriz vai ser gerada com todos os valores a 0, onde depois, através da variável *indexList*, vão ser colocados os valores armazenados em *pStar* nas respectivas colunas, de forma a criar uma matriz com o formato apresentado na Equação 2.14, que é então guardada na variável com o nome *sparsePStar*. Depois de obtida esta matriz, o último passo de cada iteração consiste em atualizar as matrizes globais, representadas na Equação 3.4.

$$\begin{aligned} sparseGroup &= [sparseGroup \quad sparsePStar^{\intercal}] \\ affineGroup &= [affineGroup \quad optimalAffine^{\intercal}] \end{aligned} \tag{3.4}$$

Quando todas as iterações terminarem, as matrizes globais vão estar preenchidas, sendo apenas necessário efetuar a sua transposição para ficarem com o formato apresentado nas Equações 2.15 e 2.16. Deste modo, já estão calculadas todas as componentes necessárias para obter a solução da equação que calcula as posições finais de cada vértice, como já foi possível ver na dedução feita na Equação 2.17. Em termos práticos, para obter esta solução foi utilizada a Pseudoinversa de *Moore–Penrose* [33], que corresponde à função *pinv* no *Matlab*. Esta decisão deveu-se a pesquisas e testes efetuados com e sem a utilização da Pseudoinversa de *Moore–Penrose*, através dos quais conclui que a sua utilização gerava resultados mais precisos. A fórmula associada à Pseudoinversa de *Moore–Penrose* está na Equação 3.5, seguida da sua aplicação na Equação 3.6, onde os resultados são armazenados na variável *vertexPositions*.

$$pinv(A) = (A^{\intercal} * A)^{-1} * A^{\intercal} \tag{3.5}$$

$$vertexPositions = pinv(sparseGroup) * affineGroup \tag{3.6}$$

A matriz de resultados, *vertexPositions*, tem dimensão Nx2, onde N é o número de vértices total (Normais+Steiner) dos polígonos e cada linha correspondente à coordenada de cada vértice. No entanto é ainda necessário aplicar a Translação a esta matriz, uma vez que o polígono resultante encontra-se centrado na origem, sendo esta componente abordada na Secção 3.3.5.

$$vertexPositions = \begin{bmatrix} v_{1x} & v_{1y} \\ v_{2x} & v_{2y} \\ \vdots & \vdots \\ v_{nx} & v_{ny} \end{bmatrix} \tag{3.7}$$

É apresentado de seguida uma síntese, em pseudocódigo, do fluxo do algoritmo descrito.

```
for each line in triangles.txt do
    [source,target] = getTrianglesFromIndex(index);
    optimalAffine = calculateOptimalTransformation(source,target,instant);
    sparsePStar = calculateSparsePVector(source,indexList);
    sparseGroup=[sparseGroup   sparsePStar^T];
    affineGroup=[affineGroup   optimalAffine^T];
end
sparseGroup = sparseGroup^T;
affineGroup = affineGroup^T;
vertexPositions = pinv(sparseGroup)*affineGroup;
vertexPositions = translateToTheNewPosition(vertexPositions, instant);
```

Algoritmo 5: Síntese do algoritmo de Interpolação Rígida Ótima

3.3.4 CÁLCULO DA MATRIZ AFINA

Na obtenção da matriz afina é utilizada uma função do *Matlab*, denominada *fitgeotrans* [34] e para a qual foi necessária uma *Toolbox* adicional, de nome Image Processing Toolbox. A sintaxe geral desta função recebe três argumentos:

- *movingPoints*: conjunto de pontos inicial que se pretendem transformar;
- *fixedPoints*: conjunto de pontos final;
- *transformationType*: tipo de transformação a ser efetuada, onde existem quatro tipos: *affine*, *nonreflectivesimilarity*, *projective* e *similarity*.

Sempre que é necessário efetuar o cálculo de uma matriz afina, que representa a transformação de um triângulo origem num destino, é utilizada a função *getTransformationMatrix*, que se encontra no ficheiro *interpolation.m*. Esta função recebe dois argumentos:

- *source*: coordenadas do triângulo origem;
- *target*: coordenadas do triângulo destino.

Nesta função é utilizado o *fitgeotrans*, com os argumentos de entrada referidos, onde os *movingPoints* correspondem às coordenadas do triângulo origem e os *fixedPoints* correspondem às coordenadas do triângulo destino, estando ambos representados por matrizes com dimensão

3x2. O tipo de transformação utilizado, representado em *transformationType*, é 'affine', ou seja:

$$tform = fitgeotrans(source,target,'affine') \tag{3.8}$$

O resultado é armazenado na variável *tform*, sendo esta uma transformação do tipo *affine2d*, que contém duas componentes:

- *tform.T*: Resultados da matriz afina calculada;
- *tform.Dimensionality*: Dimensionalidade da transformação geométrica.

Os resultados da matriz afina, Equação 3.9, têm dimensão 3x3, no entanto estes estão num formato diferente do referido na Equação 2.2. A correção das posições é obtida através de uma transposição da matriz, Equação 3.10, e que já corresponde com o formato pretendido.

$$tform.T = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{3.9}$$

$$tform.T^{\intercal} = \begin{bmatrix} a & c & t_x \\ b & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

3.3.5 INTERPOLAÇÃO DA TRANSLAÇÃO

Nas técnicas de Interpolação Rígida, a Translação é interpolada em separado das restantes componentes da transformação, sendo necessário fazer a Translação do polígono obtido para a posição correta. Para isso foi implementado um algoritmo na função *translateToTheNewPosition*, do ficheiro *interpolation.m*, que recebe dois argumentos de entrada:

- *polygonO*: coordenadas, na origem, do polígono que vai ser deslocado;
- *instant*: instante da transformação, pertencente ao intervalo [0 ; 1].

Os algoritmo está dividido em duas fases: a primeira fase consiste em fazer a translação do polígono para a sua posição original (polígono origem). A segunda fase consiste em fazer uma Interpolação Linear das distâncias entre os centroides dos polígonos origem e destino para o instante pretendido. Esta ação faz uma estimativa de onde é suposto o centroide se encontrar naquele instante, sendo apenas necessário adicionar essa translação à já efetuada anteriormente na primeira fase. A componente da Translação está representada na Equação 3.11.

$$translatedPolygon = polygonO + centroidSource + delta * instant \tag{3.11}$$
$$delta = centroidTarget - centroidSource$$

Na Figura 3.13 está representada a ideia geral do algoritmo de translação, onde a azul está representado um triângulo origem e a vermelho um triângulo destino, com os seus centroides e as respectivas fases do algoritmo também representadas.

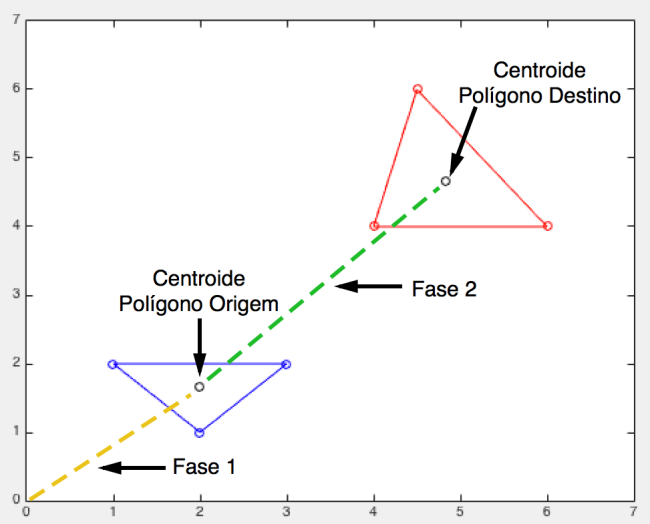


Figura 3.13: Ideia geral do algoritmo de translação

3.3.6 VARIAÇÃO DA MATRIZ AFINA ÓTIMA

Num processo de *Morphing*, onde seja usada uma técnica de Interpolação Rígida, é necessário calcular a matriz afina ótima que é composta por duas matrizes, $R_\gamma(t)$ e $S(t)$, que vão variar de acordo com um dado instante t. No entanto é necessário ter atenção à forma como é feita esta variação temporal, mais concretamente em relação à componente de rotação, uma vez que se a variação for aplicada diretamente sobre a matriz de rotação por vezes podem ser gerados resultados incorretos e que, por consequência, resultam numa transformação incorreta. Estes casos específicos acontecem sempre que é necessário fazer uma rotação de $\frac{k\pi}{2}$, onde k $\in \mathbb{Z}$, uma vez que são ângulos que resultam numa matriz de rotação com componentes de valor 0, matrizes estas representadas na Figura 3.14. Se a variação for feita diretamente, então estas componentes da matriz serão sempre 0 para qualquer instante da rotação, o que não é suposto acontecer e que vai gerar uma transformação incorreta, como é possível verificar na Figura 3.15. Este exemplo mostra as sucessivas formas de um polígono sobre o qual é aplicada uma rotação de 180º.

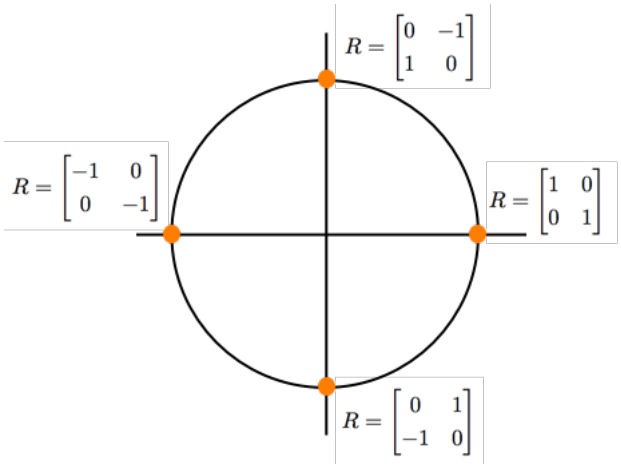


Figura 3.14: Representação das matrizes de rotação

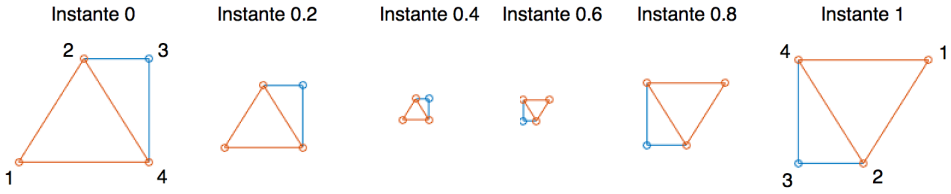


Figura 3.15: Rotação de 180º interpolada incorretamente

A solução para este problema é obtida considerando o ângulo de rotação como fator de variação em vez da matriz de rotação. Desta forma é possível saber qual o valor do ângulo de rotação para um dado instante, sendo depois necessário calcular a matriz de rotação para esse ângulo. Este processo está implementado na função *varyAffineMatrixInstant*, do ficheiro *interpolation.m*, que recebe como argumentos:

- R_γ: matriz de Rotação da transformação;
- S: matriz de Deformação (Mudança de Escala e *Shear*) da transformação;
- instant: instante da transformação.

Primeiro é necessário extrair o ângulo da matriz de rotação R_γ, para essa extração é utilizado o arco tangente de dois argumentos, função *atan2* no *Matlab* [35], que utiliza as duas componentes de rotação, $cos(\theta)$ e $sin(\theta)$, como demonstrado na Equação 3.13.

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.12}$$

$$angulo = atan2(sin(\theta), cos(\theta)) \tag{3.13}$$

Depois de extraído o ângulo de rotação, é necessário calcular o seu valor num instante, entre 0 e 1, situação representada na Equação 3.14. A matriz de rotação para o instante t, representada na Equação 3.15, é calculada tendo em conta o ângulo calculado.

$$anguloInstante = angulo * t \tag{3.14}$$

$$R_\gamma(t) = \begin{bmatrix} cos(anguloInstante) & -sin(anguloInstante) \\ sin(anguloInstante) & cos(anguloInstante) \end{bmatrix} \tag{3.15}$$

A variação da matriz S é aplicada diretamente sobre a matriz, como representado na Equação 3.16, sendo no final gerada a matriz afina ótima para o instante t, $A_\gamma(t)$, através da multiplicação das duas componentes calculadas neste processo, representada na Equação 3.17. Na Figura 3.16 é possível ver, para o mesmo exemplo utilizado anteriormente, os resultados depois da aplicação deste algoritmo, onde é clara a sua melhoria relativamente ao exemplo anterior.

$$\begin{aligned} & S(t) = (1-t) * I + t * S \\ \Leftrightarrow\ & S(t) = (1-t) * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + t * \begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \end{aligned} \tag{3.16}$$

$$A_\gamma(t) = R_\gamma(t) * S(t) \tag{3.17}$$

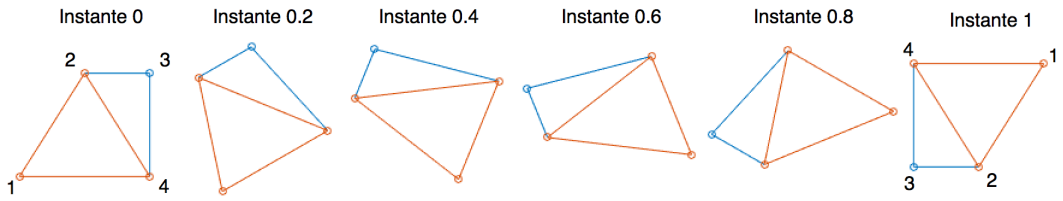


Figura 3.16: Rotação de 180º interpolada corretamente

3.3.7 CONSISTÊNCIA DAS ROTAÇÕES

A Consistência de Rotações implementada teve como base a técnica proposta por *Baxter et al* [25], descrita na Secção 2.3.5. Para a implementação da técnica proposta foi necessário implementar algumas funcionalidades, nomeadamente:

- Estrutura de dados do tipo *Stack*;

- Obtenção da lista de triângulos fronteira;
- Obtenção da lista de triângulos vizinhos de um outro;
- deteção de descontinuidades e cálculo dos novos ângulos;
- Minimização das rotações.

No *Matlab* não existe uma estrutura do tipo *Stack*, pelo que foram desenvolvidas duas funções para fazer uma matriz comportar-se como uma Stack. Estas duas funções consistem em implementações das funcionalidades de *Push* e *Pop*, que são utilizadas para adicionar e remover elementos, respetivamente, e cujo funcionamento está representado na Figura 3.17. Nesta Figura, inicialmente é apresentada uma *Stack* com quatro elementos, sendo feito o *Push* do elemento 220, elemento este que é adicionado no topo da *Stack*. Quando é feito um *Pop*, é retirado o elemento no topo da *Stack*, que, neste exemplo, é o elemento 220. Estas funções estão implementadas no ficheiro *interpolation.m* com o nome:

- *stackPush*: representa o *Push* para a *Stack*. Recebe como argumentos a matriz e o elemento a adicionar, que vai ser colocado na primeira posição. Retorna a matriz atualizada.
- *stackPop*: representa o *Pop* da *Stack*. Recebe como argumento a matriz à qual vai ser retirado o primeiro elemento. Retorna a matriz atualizada e o valor retirado.

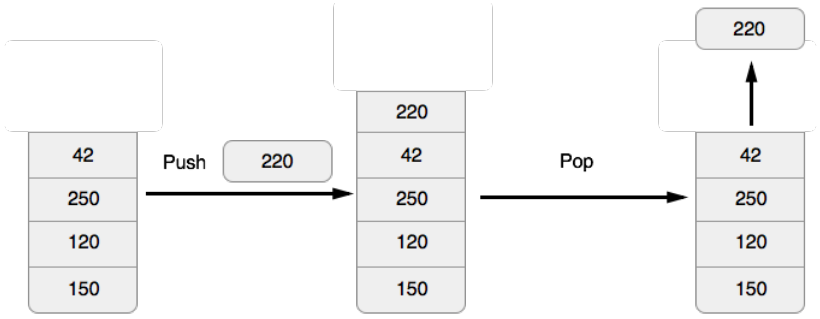


Figura 3.17: Exemplo de *Push* e *Pop*

Em relação à obtenção da lista de triângulos fronteira. O algoritmo está implementado na função *getBoundaryTriangles*, do ficheiro *interpolation.m*. Esta consiste em fazer uma pesquisa na lista de triângulos, que se encontra no ficheiro *triangles.txt*, onde para cada triângulo da lista é verificado se, pelo menos, dois dos seus vértices pertencem à lista inicial de vértices. Em caso afirmativo, esse triângulo é considerado como sendo da fronteira. A Figura 3.18 apresenta um exemplo de um polígono onde a verde estão representados os triângulos considerados fronteira e a vermelho os não considerados fronteira.

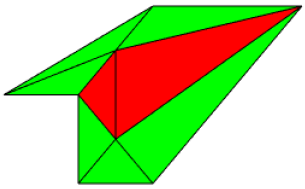


Figura 3.18: Triângulos Fronteira

A lista de triângulos vizinhos é obtida de forma similar à dos triângulos fronteira. O algoritmo está implementado na função *getNeighborTriangles*, do ficheiro *interpolation.m*, que recebe como argumento o triângulo sobre o qual vão ser calculados os vizinhos. Esta função consiste em analisar a lista de triângulos, que se encontra no ficheiro *triangles.txt*, à procura de quais partilham pelo menos dois vértices com o triângulo passado como argumento. Sempre que for encontrado um triângulo com pelo menos dois vértices partilhados, é adicionado à lista de vizinhos. A Figura 3.19 apresenta um exemplo onde a azul está representado o triângulo sobre o qual são calculados os vizinhos e a verde estão representados aqueles que são considerados seus vizinhos.

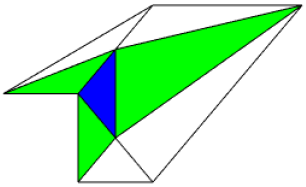
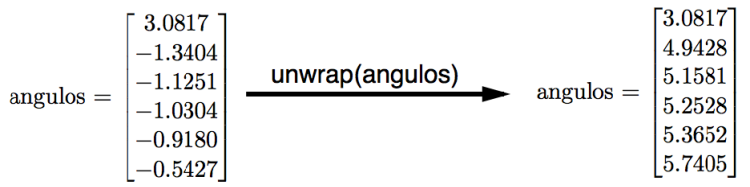
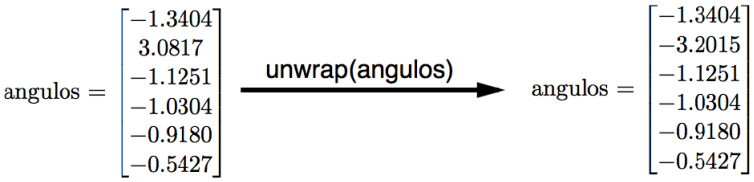


Figura 3.19: Triângulos Vizinhos

Considera-se que existe uma descontinuidade quando as diferenças entre as rotações de dois triângulos vizinhos ultrapassam os 180º. Quando esta situação ocorre são adicionados ou subtraídos, dependendo da situação, múltiplos de 360º para colocar a diferença entre os ângulos abaixo dos 180º. No *Matlab* já existe uma função que faz precisamente o que é necessário, denominada *unwrap* [36]. A sintaxe desta função recebe apenas um argumento, que é a lista de ângulos de rotação, em radianos, para o qual vai ser verificado se existem descontinuidades e, caso existam, será feito o ajuste do ângulo de rotação em questão de forma a remover a descontinuidade. É importante referir que o primeiro valor na lista dos ângulos tem elevada importância, uma vez que é este que vai definir o fluxo que os seguintes vão seguir, como é possível de verificar na Figura 3.20. Esta Figura mostra um exemplo da aplicação do *unwrap* em dois vectores com ângulos de rotação iguais, apenas com os dois primeiros elementos trocados para que seja percetível a diferença de resultados gerada pela alteração do primeiro elemento da lista. Na Figura 3.20a é possível verificar que foi necessário ajustar todos os ângulos de rotação, enquanto que na Figura 3.20b apenas foi necessário ajustar um ângulo de rotação. Os resultados finais em ambos os exemplos vão ser os mesmos, no entanto o primeiro exemplo vai gerar um excesso de rotação, situação demonstrada anteriormente na Figura 2.20, que é desnecessária e acima de tudo indesejável.



(a) Primeiro elemento prejudicial



(b) Primeiro elemento benéfico

Figura 3.20: Influência do primeiro elemento no *unwrap*

De modo a resolver os possíveis excessos de rotação, foi implementado o algoritmo proposto por *Baxter et al* [25], para que o resultado final não seja dependente da escolha do primeiro ângulo de rotação na lista. O algoritmo está implementado na função *minimizeRotations*, do ficheiro *interpolation.m*, que recebe como argumento a lista de ângulos de rotação a ser minimizada. O seu funcionamento consiste em fazer a média dos vários ângulos de rotação da lista, aos quais vão ser adicionados ou subtraídos, dependendo se a média calculada for positiva ou negativa, múltiplos de 360º até que a média fique o mais próxima possível de 0. Quando o valor absoluto da média for mínimo, então o excesso de rotação foi eliminado. A Figura 3.21 apresenta os resultados aplicados sobre o exemplo anterior, onde agora, independentemente do primeiro valor da lista de ângulos, o resultado final será sempre o ideal.

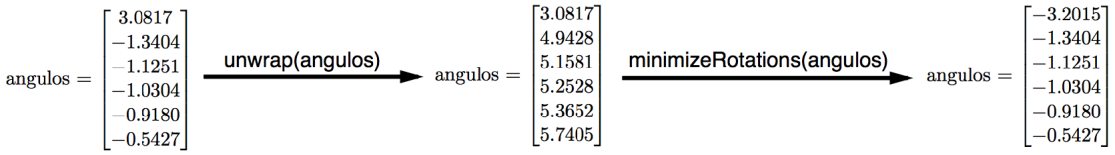


Figura 3.21: Rotações minimizadas

Com os algoritmos referidos, é possível agora integrar a consistência de rotações com os dois tipos de Interpolação Rígida implementados, seguindo o fluxo da técnica proposta por *Baxter et al* [25]. Para ambas as implementações, esta integração é dividida em duas fases: a primeira fase é utilizada para calcular as novas matrizes de rotação e as restantes componentes, já abordadas anteriormente, utilizadas para gerar as transformações. A análise dos vários triângulos segue o critério de primeiro serem analisados os triângulos fronteira, para que estes possam propagar as boas rotações para os seus vizinhos. Foi desenvolvida uma função denominada *calculateInterpolationComponents*, no ficheiro *interpolation.m*, que recebe como argumento uma lista de triângulos que é utilizada para calcular as várias componentes utilizadas para a transformação desse triângulo, nomeadamente:

- *angles*: Matriz com os ângulos de rotação dos triângulos analisados, que vai ser utilizada para procurar descontinuidades;
- *pStar*: Matriz utilizada na técnica descrita na Secção 3.3.3
- S: Matriz que representa a componente de deformação (Mudança de Escala e *Shear*) da transformação.

Cada uma destas componentes vai ser armazenada numa matriz global que vai conter as várias componentes para todos os triângulos. Depois de geradas as matrizes globais, vai ser gerada a matriz global das rotações, sob as quais já foram removidas as descontinuidades, através da função *calculateRotationMatrixGroup*. Esta função recebe como argumento a lista dos novos ângulos calculados, no entanto é necessário fazer a verificação se algum destes ângulos é maior que $2*\pi$ ou menor que $-2*\pi$. Em caso afirmativo é necessário ignorar os novos ângulos porque a rotação vai gerar sempre interseções devido ao facto de ser maior que uma volta completa. Nestas situações é necessário utilizar os ângulos iniciais, que também foram guardados ao longo da análise dos triângulos. Por último, vai ser calculada a matriz global que vai armazenar todas as matrizes afinas, sendo para isso utilizada a função *varyAffineTransformationInstant* que recebe a matriz global de rotações, a matriz global da deformação e o instante da transformação. Depois de obtida a matriz afina global, os algoritmos de Interpolação Rígida seguem o seu fluxo natural, de acordo com o que foi descrito anteriormente nas subsecções 3.3.2 e 3.3.3.

3.3.8 FICHEIROS UTILIZADOS

No algoritmo desenvolvido para o *Morphing* são utilizados vários ficheiros, que se encontram na pasta *Files*, sendo aqui feito um breve resumo dos objetivos de cada um deles:

- *vertexPositions.txt*
  - Ficheiro utilizado na Interpolação Rígida Média para guardar as coordenadas que todos os vértices partilhados têm ao longo da transformação, para que no final possa ser feita a média e obtida a coordenada final de cada um deles.
- *auxiliar.txt*
  - Ficheiro utilizado como auxílio quando é necessário apagar alguma linha do ficheiro *vertexPositions.txt*.

3.4 INTERFACE GRÁFICA

Todas as ferramentas implementadas estão acessíveis através de uma interface gráfica, resultando assim na aplicação final, denominada *T-Morph*. Esta interface foi desenvolvida

através do GUIDE (*Graphical User Interface Design Environment*), que é um conjunto de ferramentas, em *Matlab*, utilizadas para o desenvolvimento de interfaces gráficas. Na interface gráfica são utilizados dois ficheiros:

- *TMorph.fig*:
  - Ficheiro onde é guardado o *Layout* da interface criada através do GUIDE.
- *TMorph.m*:
  - Ficheiro com as *callbacks* de cada componente, onde é programado o comportamento que cada uma vai ter.

Na Figura 3.22 é possível ver o resultado final da interface desenvolvida.

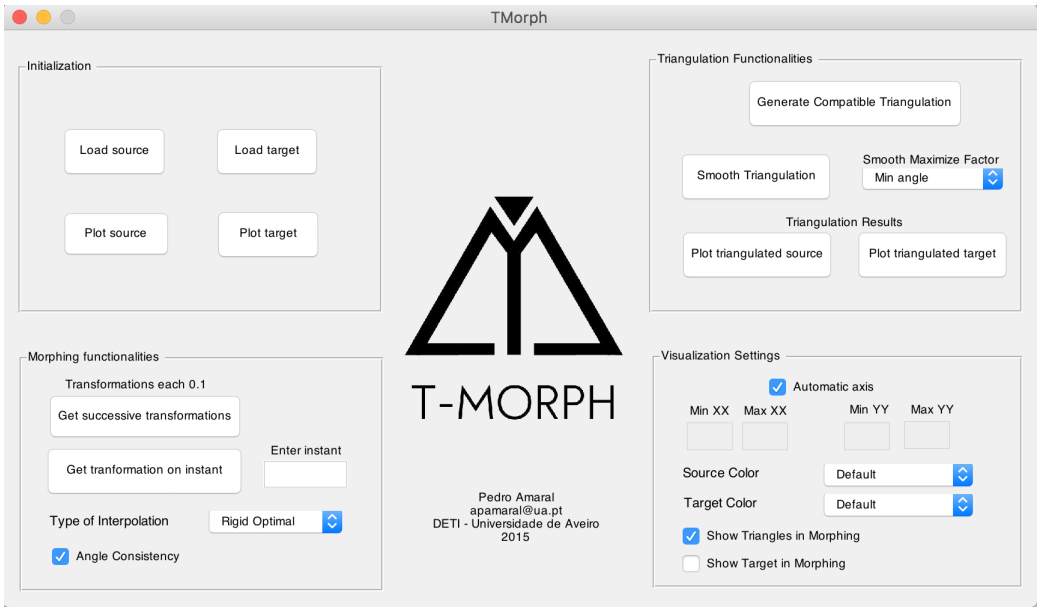
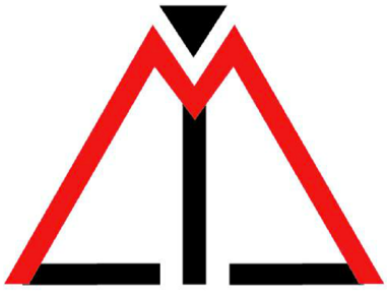


Figura 3.22: Interface Gráfica

O nome *T-Morph* é derivado das duas componentes principais estudadas nesta Dissertação: Triangulação e *Morphing*. Também associado a estas componentes está o logótipo da aplicação, onde, através da Figura 3.23a, é possível verificar que este tem a forma de um Triângulo, juntamente com as iniciais T de Triangulação e M de *Morphing*, representadas, respetivamente, nas Figuras 3.23b e 3.23c.

(a) Representação do triângulo

(b) Representação do T



(c) Representação do M

Figura 3.23: Explicação do Logótipo

Em relação à interface, esta foi dividida em quatro painéis, cada um destes utilizado para agrupar funcionalidades do mesmo tipo. O primeiro painel, apresentado na Figura 3.24, é utilizado para as funcionalidades de inicialização, neste caso para as leituras, através de ficheiros, dos polígonos origem e destino.

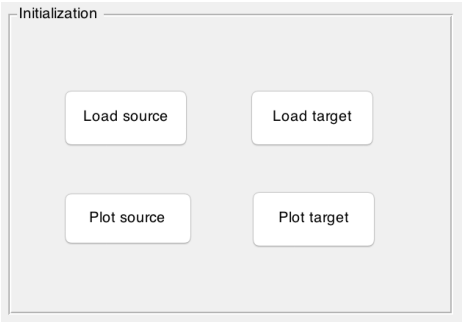


Figura 3.24: Painel de leitura dos polígonos

O segundo painel, apresentado na Figura 3.25, agrupa as funcionalidades relativas à Triangulação, mais concretamente a geração de Triangulações Compatíveis, através do botão *Generate Compatible Triangulation*, e a Suavização da malha de triângulos, através do botão *Smooth Triangulation*, onde é ainda possível escolher o fator de melhoria: maximização dos ângulos mínimos ou das áreas mínimas.

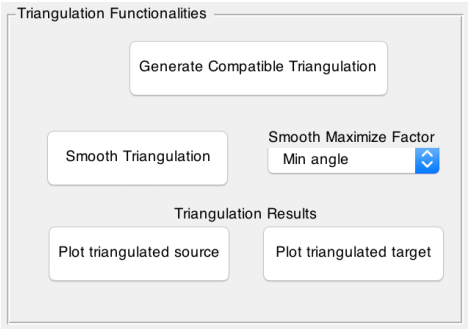


Figura 3.25: Painel das funcionalidades de Triangulação

O terceiro painel, apresentado na Figura 3.26, é composto pelas funcionalidades de *Morphing*, onde a principal é a obtenção das transformações intermédias do polígono origem até ao polígono destino, acessível através do botão *Get successive transformations*, com transformações a cada 0.1. Outra funcionalidade é gerar a transformação para um dado instante, através do botão *Get transformation on instant*, onde é possível introduzir qualquer instante entre [0 ; 1] na caixa de texto. Por último, é possível definir o tipo de interpolação a utilizar na transformação, dentro dos três tipos referidos nas Secções 3.3.1, 3.3.2 e 3.3.3, existindo ainda a possibilidade de ativar a consistência de rotações, abordada na Secção 3.3.7.

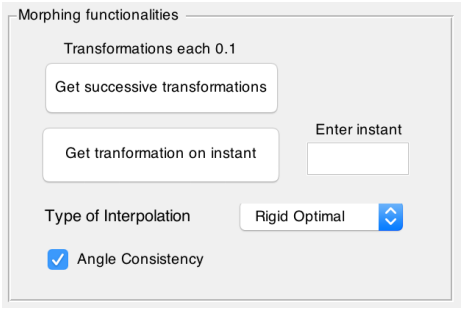


Figura 3.26: Painel das funcionalidades de *Morphing*

O quatro painel, apresentado na Figura 3.27, é utilizado para agrupar as definições de visualização dos resultados, onde podem ser alterados os valores dos eixos nas janelas de visualização, alterar as cores de visualização dos polígonos e opções para ativar ou desativar a visualização permanente dos triângulos e do polígono destino ao longo das várias transformações no processo de *Morphing*.

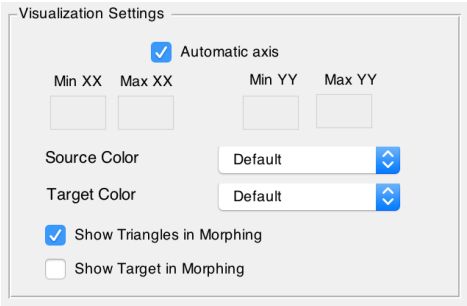


Figura 3.27: Painel das funcionalidades de Visualização

3.5 SÍNTESE DO CAPÍTULO

Este capítulo teve como objetivo fazer uma descrição dos algoritmos implementados para a resolução do problema proposto.

Sobre a Triangulação Compatível, foi implementada uma solução para a geração da malha de triângulos, tendo em conta o algoritmo proposto por *Surazhsky et al* [15]. Além desse algoritmo foi ainda implementado outro para fazer uma melhoria da malha de triângulos.

Em relação ao *Morphing*, foram implementadas três soluções distintas para a resolução do *Vertex Path Problem*, uma através de Interpolação Linear e duas através de Interpolação Rígida, tendo sido utilizados os conceitos estudados por *Alexa et al* [6] e *Baxter et al* e [25].

Por último foi desenvolvida uma interface gráfica, com recurso ao conjunto de ferramentas GUIDE, para fazer a interligação entre todas as componentes.

capítulo 4

Resultados

4.1 introdução

Este capítulo apresenta os resultados obtidos pelos algoritmos implementados, com o objetivo de os avaliar, comparar e demonstrar a viabilidade da solução apresentada.
A Secção 4.2 apresenta os resultados utilizando um conjunto de polígonos preparados especificamente para estes testes. Na Triangulação Compatível, o foco da análise é a malha de triângulos resultante, onde vão ser comparados os resultados obtidos da triangulação inicial com os obtidos depois de melhorar a malha com recurso à Suavização (sempre que tal seja possível). Sobre o *Morphing*, são comparadas as três técnicas de interpolação implementadas. Os principais aspetos de comparação são a qualidade com que a transformação decorre e a ocorrência de interseções.
A Secção 4.3 apresenta os resultados utilizando um conjunto de dados reais, que representam o movimento de *Icebergs*. Nestes testes, o objetivo é estimar o movimento que os *Icebergs* tiveram, usando as técnicas implementadas, e comparar a similaridade desses resultados com aquilo que realmente aconteceu.

Para os testes desenvolvidos, o computador utilizado foi um Macbook Pro de 2013, com um processador Intel Core i5 de 2.4 GHz e 4 GB de memória RAM.

4.2 exemplos de teste

Nesta secção são apresentados os resultados utilizando um conjunto de polígonos sintéticos com o objetivo de testar a solução implementada. Este conjunto é apresentado na Figura 4.1,

onde também estão denotadas as correspondências entre os vértices que são representadas pelos índices numéricos dos mesmos. Em relação ao último par de polígonos, apenas estão representados quais são os dois primeiros índices numéricos para demonstrar a ordem pela qual segue a numeração, caso contrário ficaria demasiado confuso de visualizar devido ao elevado número de vértices existente.

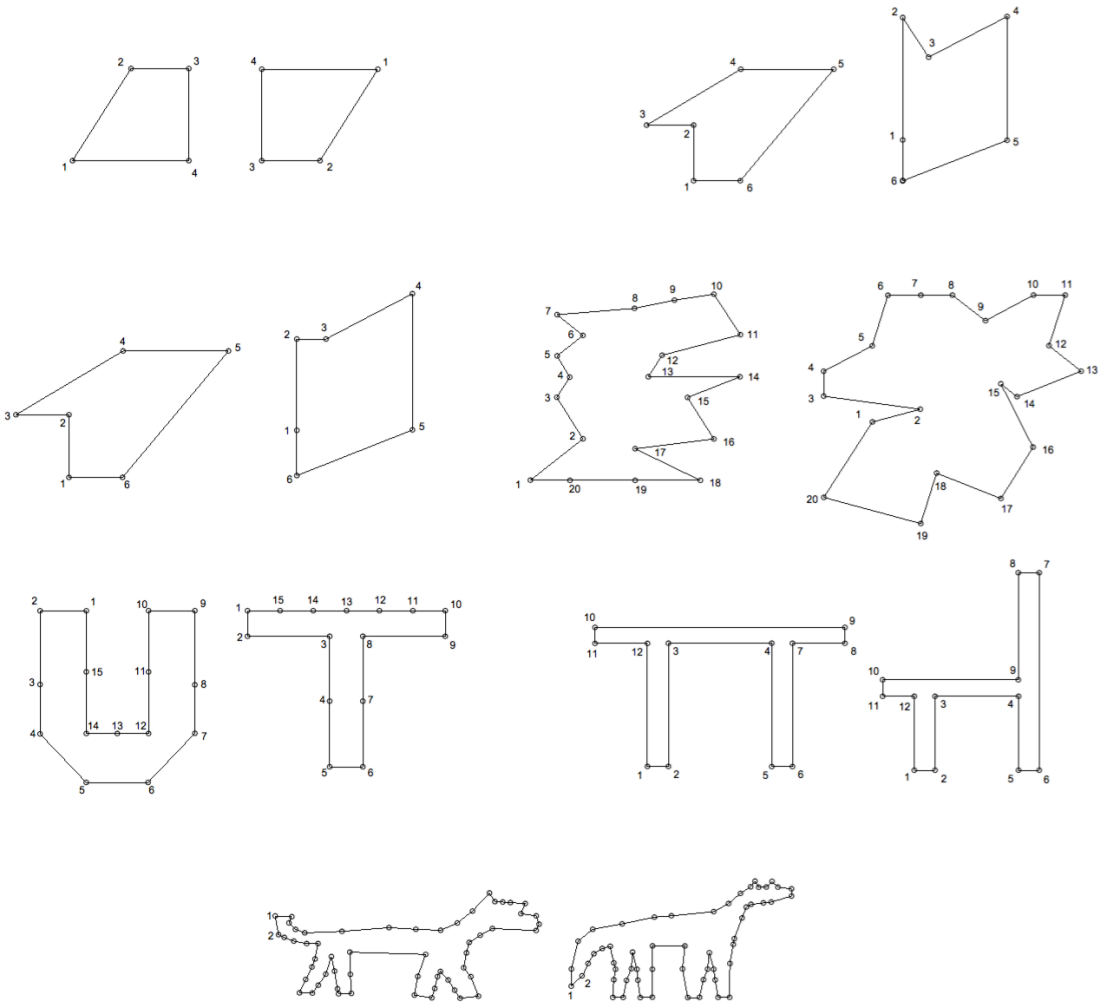


Figura 4.1: Conjunto de polígonos sintéticos

Na Triangulação Compatível, para cada um dos pares de polígonos, são apresentados os resultados da malha de triângulos e, caso tenham sido adicionados Vértices de Steiner, os resultados da melhoria da malha, tanto para maximização dos ângulos mínimos como para a maximização das áreas mínimas.

Em relação ao *Morphing*, para cada um dos pares de polígonos, é utilizada a malha de triângulos resultante da Triangulação Compatível com uma Suavização que maximiza os ângulos mínimos. São apresentados os resultados para os três tipos de interpolação implementadas: Linear, Rígida Média e Rígida Ótima, onde para as Interpolações Rígidas são

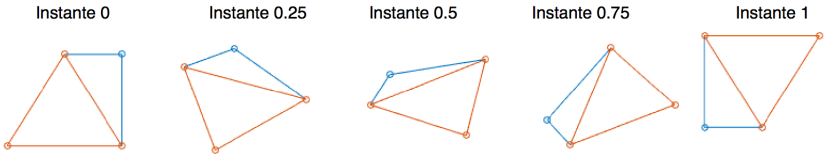
apresentados os resultados com Consistência de Rotações.

Por questões de visualização, os triângulos são desenhados com várias cores, as transformações efetuadas com intervalos de 0.25 e as correspondências de vértices ocultadas, sendo apenas visíveis na Figura 4.1, apresentada anteriormente.

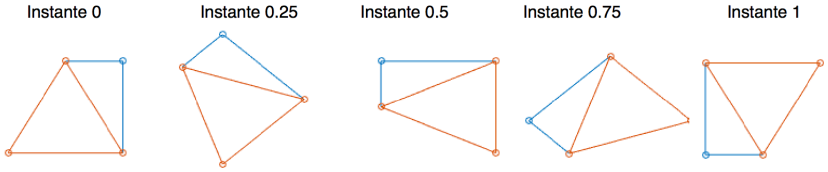
(a) Polígono Origem (b) Polígono Destino

Figura 4.2: Exemplo 1: Triangulação Compatível

(a) Interpolação Linear



(b) Interpolação Rígida Média

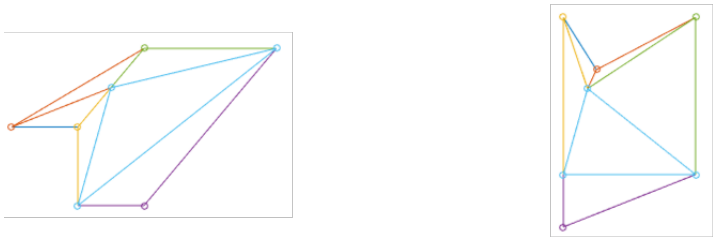


(c) Interpolação Rígida Ótima

Figura 4.3: Exemplo 1: Morphing

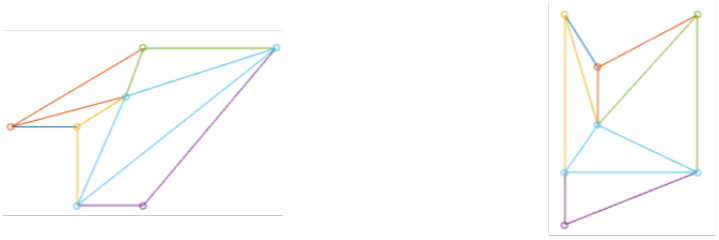
As Figuras 4.2 e 4.3 apresentam um exemplo de um polígono sobre o qual foi aplicada uma rotação de 180º. Este é um exemplo simples, não existindo muito a salientar em relação aos resultados da Triangulação. O principal objetivo para este exemplo é verificar o processo de *Morphing*, uma vez que uma rotação de 180º foi um dos casos problemáticos que foram referidos na Secção 3.3.6. É possível verificar que na Interpolação Linear os vértices convergem para o ponto central, o que é um resultado esperado tendo em conta o comportamento de uma Interpolação Linear, mas não é um resultado desejado em termos de representação de uma transformação realista. Esta situação é corrigida com os métodos de Interpolação Rígida, através dos quais é obtida a transformação esperada. Neste exemplo estão também realçadas

as diferenças entre as duas técnicas de Interpolação Rígida, onde na Interpolação Rígida Ótima os triângulos possuem uma maior área e menor variação ao longo da transformação.



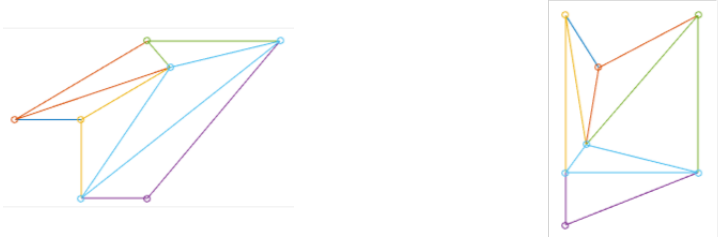
(a) Polígono Origem (b) Polígono Destino

Figura 4.4: Exemplo 2: Triangulação Compatível



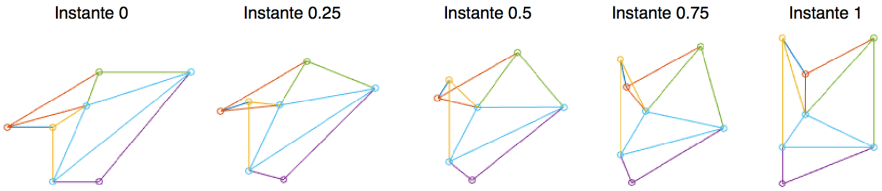
(a) Polígono Origem (b) Polígono Destino

Figura 4.5: Exemplo 2: Melhoria com maximização do ângulo mínimo

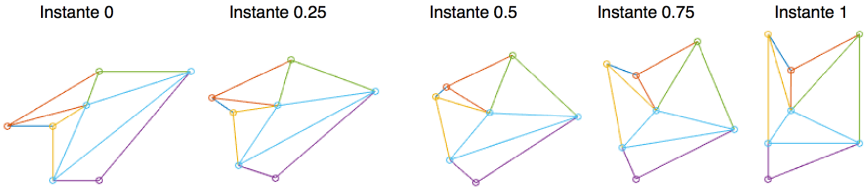


(a) Polígono Origem (b) Polígono Destino

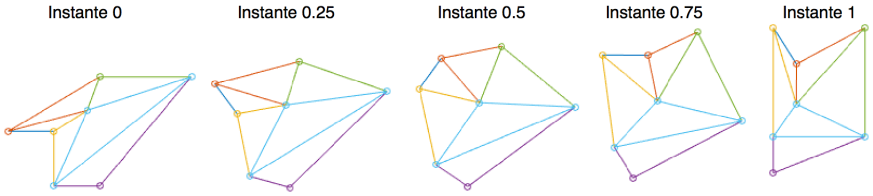
Figura 4.6: Exemplo 2: Melhoria com maximização da área mínima



(a) Interpolação Linear



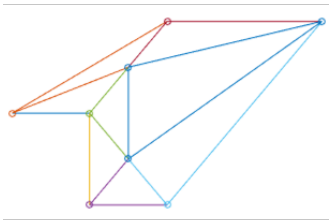
(b) Interpolação Rígida Média



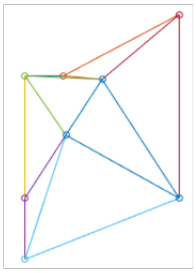
(c) Interpolação Rígida Ótima

Figura 4.7: Exemplo 2: Morphing

As Figuras 4.4, 4.5, 4.6 e 4.7 apresentam um exemplo considerado complexo, devido à utilização de polígonos côncavos que implicam uma elevada rotação para executar a transformação sem interseções. O primeiro fator a salientar é a adição de um Vértice de Steiner para gerar a Triangulação Compatível. É possível verificar que o Vértice de Steiner adicionado no polígono destino (Figura 4.4b) não está na melhor posição, situação que é corrigida com uma Suavização da malha, onde ambos os fatores de maximização geram melhores resultados. Em relação ao *Morphing*, a região côncava do polígono origem é a zona crítica da transformação, onde a Interpolação Linear gera interseções, sendo esta situação resolvida com as técnicas de Interpolação Rígida.

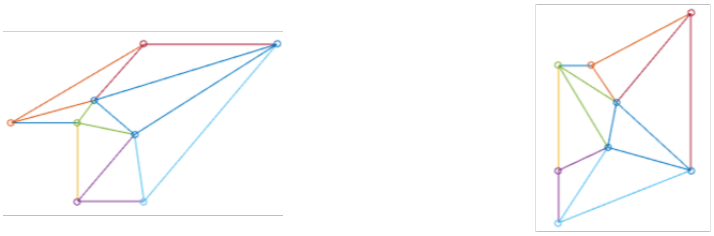


(a) Polígono Origem



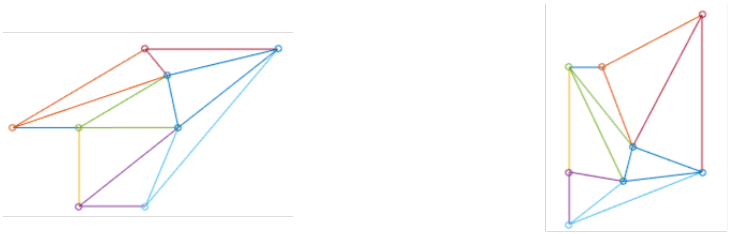
(b) Polígono Destino

Figura 4.8: Exemplo 3: Triangulação Compatível



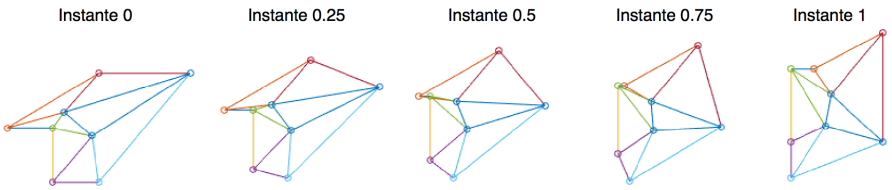
(a) Polígono Origem (b) Polígono Destino

Figura 4.9: Exemplo 3: Melhoria com maximização do ângulo mínimo

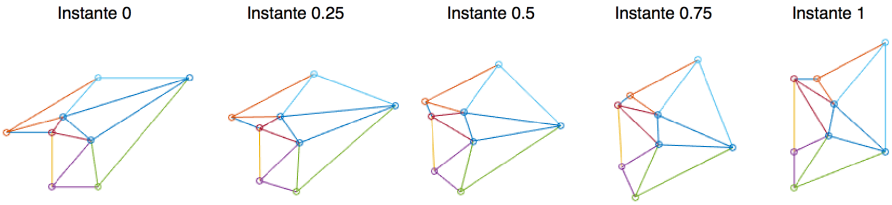


(a) Polígono Origem (b) Polígono Destino

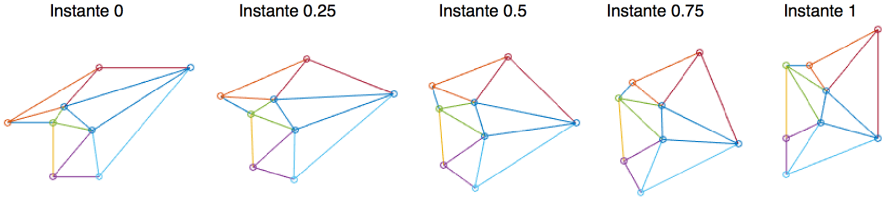
Figura 4.10: Exemplo 3: Melhoria com maximização da área mínima



(a) Interpolação Linear



(b) Interpolação Rígida Média

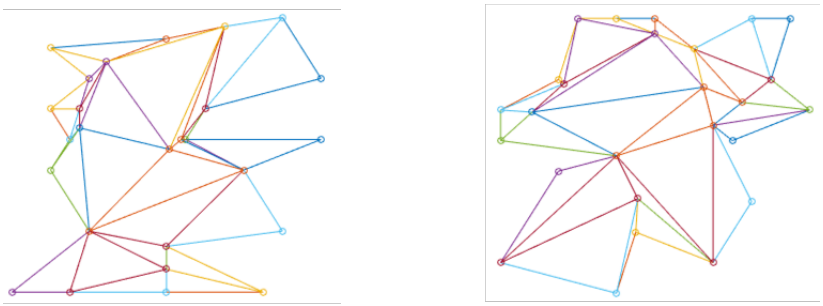


(c) Interpolação Rígida Ótima

Figura 4.11: Exemplo 3: Morphing

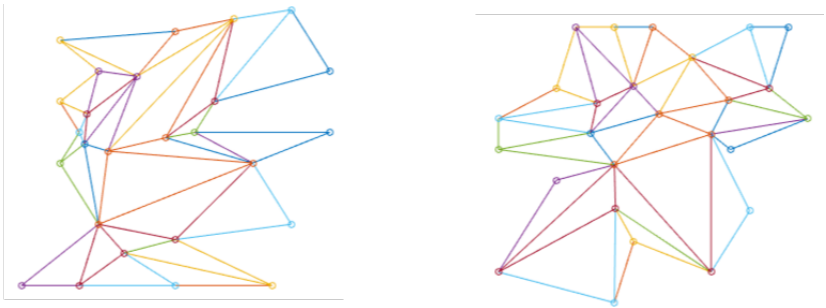
O exemplo apresentado nas Figuras 4.8, 4.9, 4.10 e 4.11 é similar ao anterior, existindo pequenas alterações no polígono destino. O foco deste exemplo está na componente de

Triangulação para demonstrar que por vezes são geradas malhas de triângulos com baixa qualidade, como é o caso da malha do polígono destino, Figura 4.8b, onde, por exemplo, o triângulo que une os vértices 2 e 3 com o Vértice de Steiner superior é tão fino que é difícil de notar a sua existência. No entanto esta situação é resolvida depois de efetuada uma Suavização da malha, com destaque para a maximização dos ângulos mínimos que gera uma malha mais regular. Em relação ao *Morphing*, o comportamento é idêntico ao exemplo anterior.



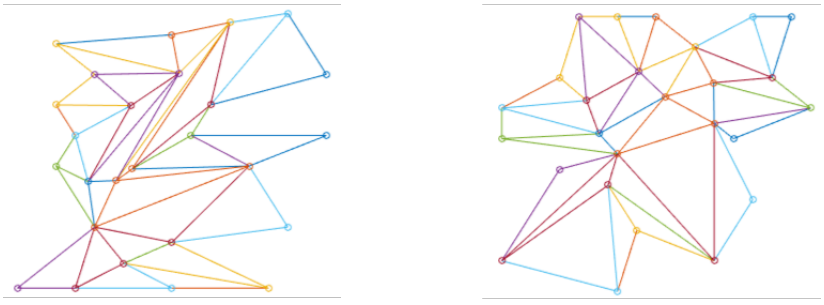
(a) Polígono Origem (b) Polígono Destino

Figura 4.12: Exemplo 4: Triangulação Compatível



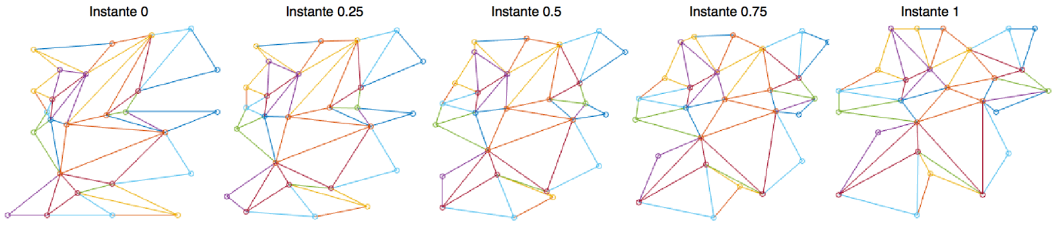
(a) Polígono Origem (b) Polígono Destino

Figura 4.13: Exemplo 4: Melhoria com maximização do ângulo mínimo

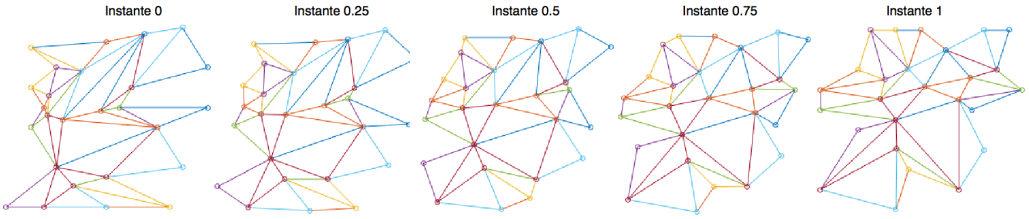


(a) Polígono Origem (b) Polígono Destino

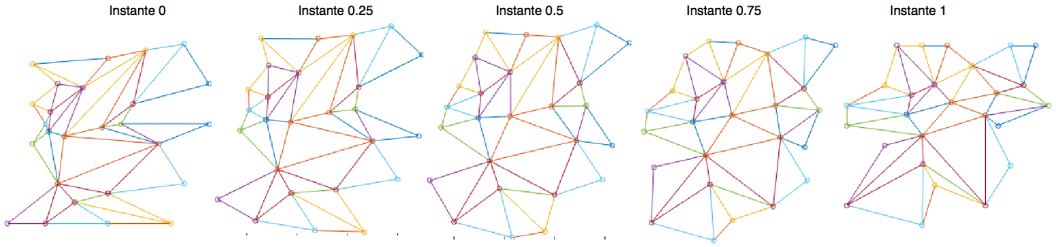
Figura 4.14: Exemplo 4: Melhoria com maximização da área mínima



(a) Interpolação Linear



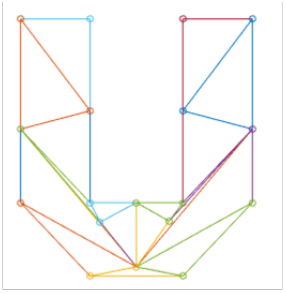
(b) Interpolação Rígida Média



(c) Interpolação Rígida Ótima

Figura 4.15: Exemplo 4: Morphing

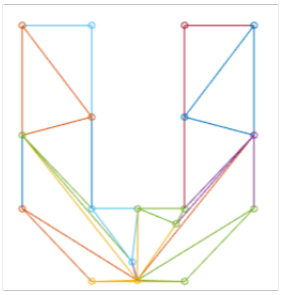
As Figuras 4.12, 4.13, 4.14 e 4.15 apresentam um exemplo com dois polígonos irregulares. Para conseguir gerar uma Triangulação Compatível foi necessário adicionar quatro Vértices de Steiner, resultando numa malha de triângulos com fraca qualidade, situação que é resolvida através da Suavização. Este é um bom exemplo para verificar diferenças de resultados entre as duas técnicas de Suavização, onde a maximização das áreas mínimas gera triângulos mais finos e longos no polígono origem, Figura 4.14a. Esta situação não acontece com a maximização dos ângulos mínimos, Figura 4.13a. Em relação ao *Morphing*, este é um exemplo onde nenhuma das técnicas de Interpolação falhou. No entanto é importante referir a quase existência de interseções na Interpolação Linear, onde existem alguns triângulos que ficam com áreas muito baixas ao longo da transformação, situação novamente corrigida com as técnicas de Interpolação Rígida.



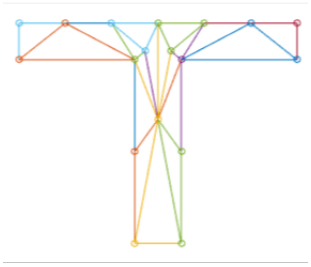
(a) Polígono Origem

(b) Polígono Destino

Figura 4.16: Exemplo 5: Triangulação Compatível

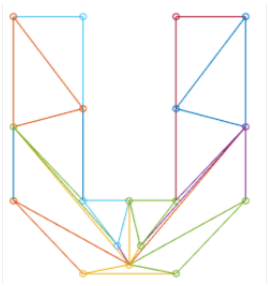


(a) Polígono Origem

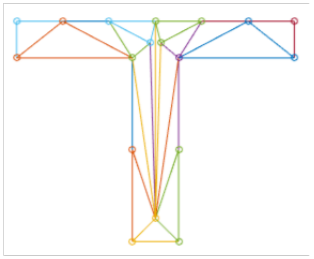


(b) Polígono Destino

Figura 4.17: Exemplo 5: Melhoria com maximização do ângulo mínimo

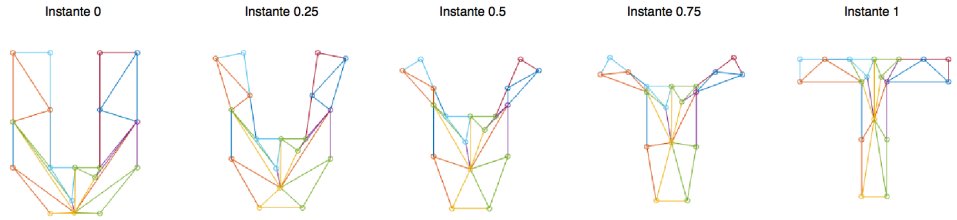


(a) Polígono Origem

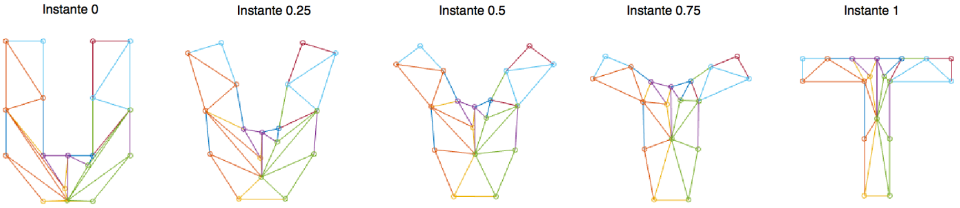


(b) Polígono Destino

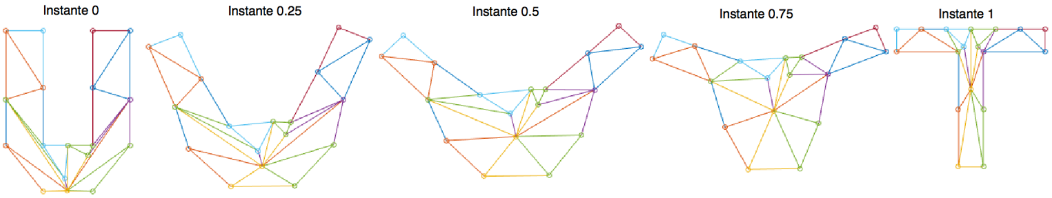
Figura 4.18: Exemplo 5: Melhoria com maximização da área mínima



(a) Interpolação Linear



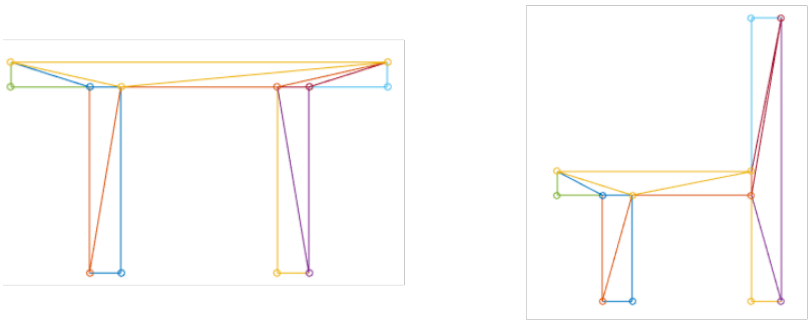
(b) Interpolação Rígida Média



(c) Interpolação Rígida Ótima

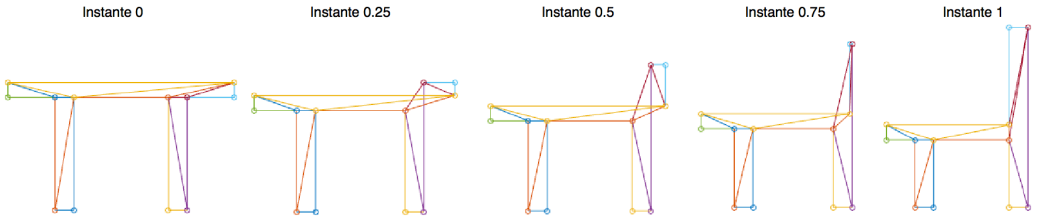
Figura 4.19: Exemplo 5: Morphing

As Figuras 4.16, 4.17, 4.18 e 4.19 apresentam um exemplo com o objetivo de fazer a transformação de polígonos que representam duas letras: U e T. A Triangulação Compatível foi obtida com recurso a três Vértices de Steiner. No entanto as malhas resultantes do polígono origem não ficaram com boa qualidade. Mesmo depois da aplicação da Suavização continuam a existir triângulos finos e longos, situação que não é possível de evitar. A fraca qualidade da malha resultante tem influência principalmente nos resultados da Interpolação Rígida Ótima, Figura 4.19c, onde é possível verificar a existência de uma deformação ao longo da transformação. Uma vez que existem triângulos finos e longos, os vértices têm de efetuar um movimento de rotação maior para que, nas transformações intermédias, seja compensada a área desses triângulos. Esta deformação é menor na Interpolação Rígida Média, uma vez que o movimento em rotação que esta faz é menor.

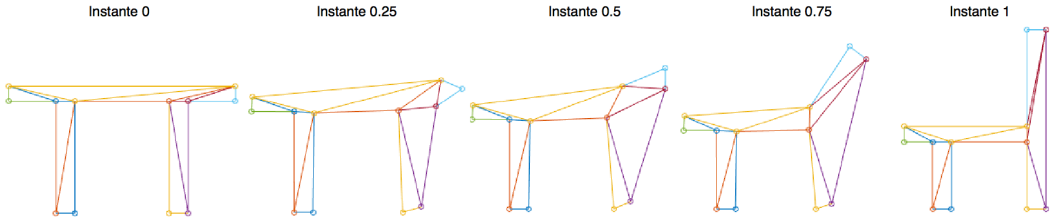


(a) Polígono Origem (b) Polígono Destino

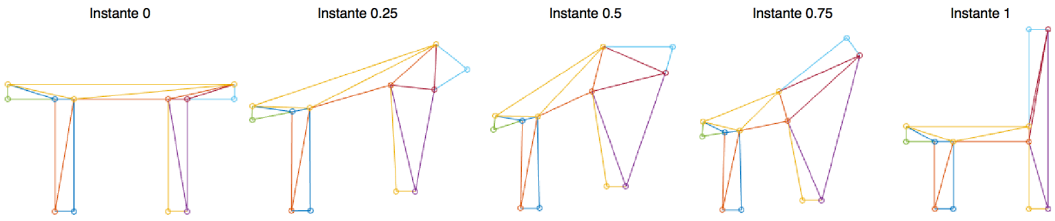
Figura 4.20: Exemplo 6: Triangulação Compatível



(a) Interpolação Linear



(b) Interpolação Rígida Média

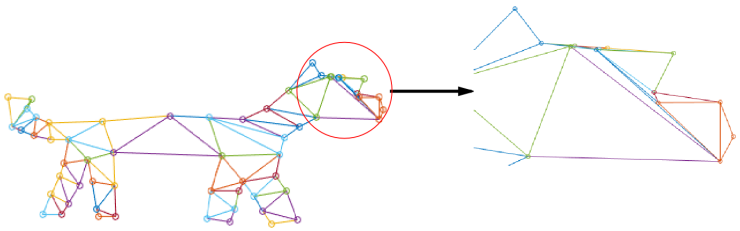


(c) Interpolação Rígida Ótima

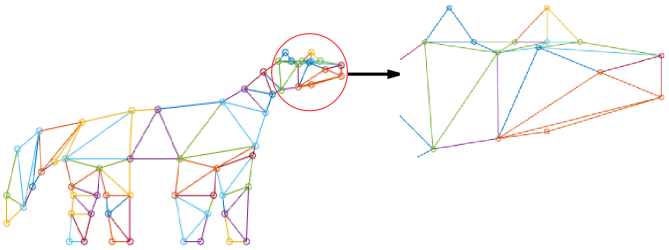
Figura 4.21: Exemplo 6: Morphing

As Figuras 4.20 e 4.21 apresentam um exemplo de uma transformação de uma mesa numa cadeira. A Triangulação Compatível foi obtida sem a adição de Vértices de Steiner, no entanto a qualidade não é a melhor e, uma vez que não existem Vértices de Steiner, não é possível de melhorar com a Suavização. Este fator influência novamente o *Morphing* onde, pelas mesmas razões do exemplo anterior, existe uma deformação criada para compensar a área dos triângulos finos e longos, compensação esta que não é tão elevada na Interpolação Rígida Média. Outro pormenor importante a referir sobre este exemplo é a transformação

sem interseções obtida pelas técnicas de Interpolação Rígida, que ocorreram quando utilizada a Interpolação Linear.

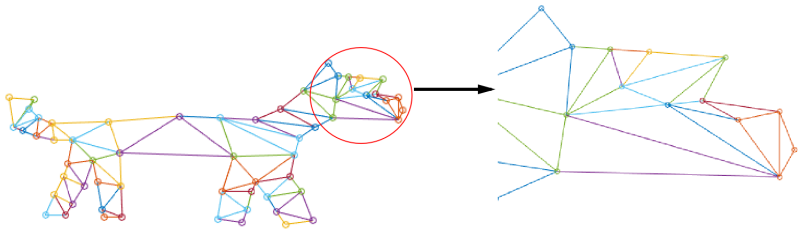


(a) Polígono Origem

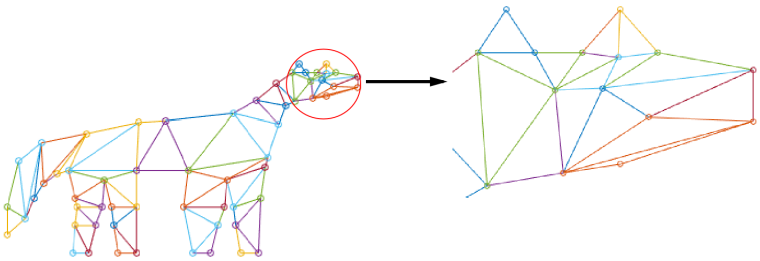


(b) Polígono Destino

Figura 4.22: Exemplo 7: Triangulação Compatível

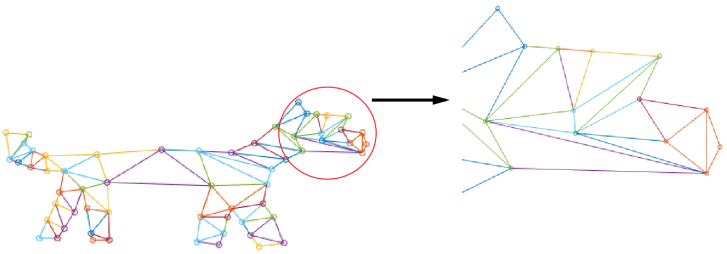


(a) Polígono Origem

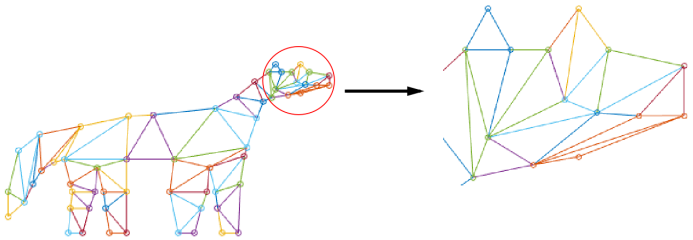


(b) Polígono Destino

Figura 4.23: Exemplo 7: Melhoria com maximização do ângulo mínimo

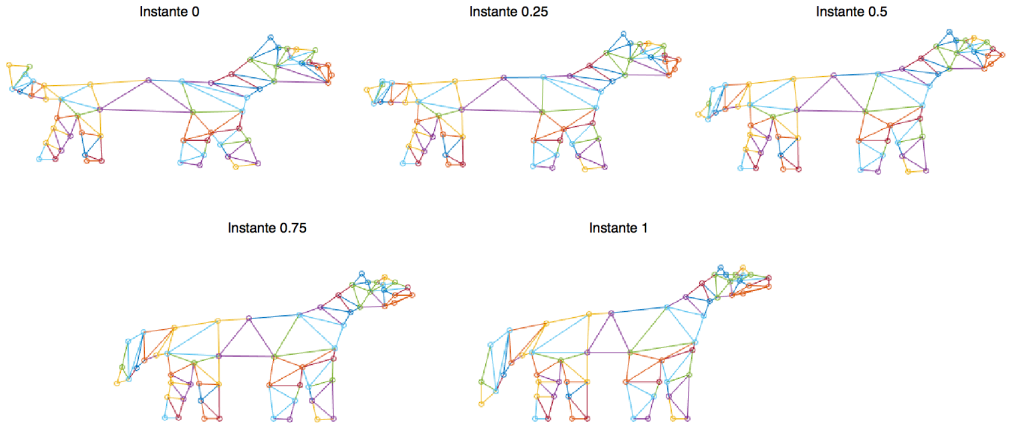


(a) Polígono Origem

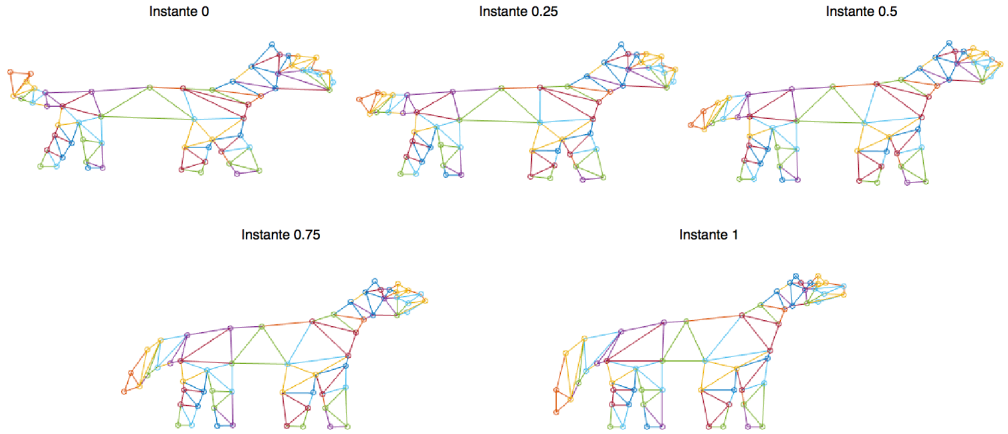


(b) Polígono Destino

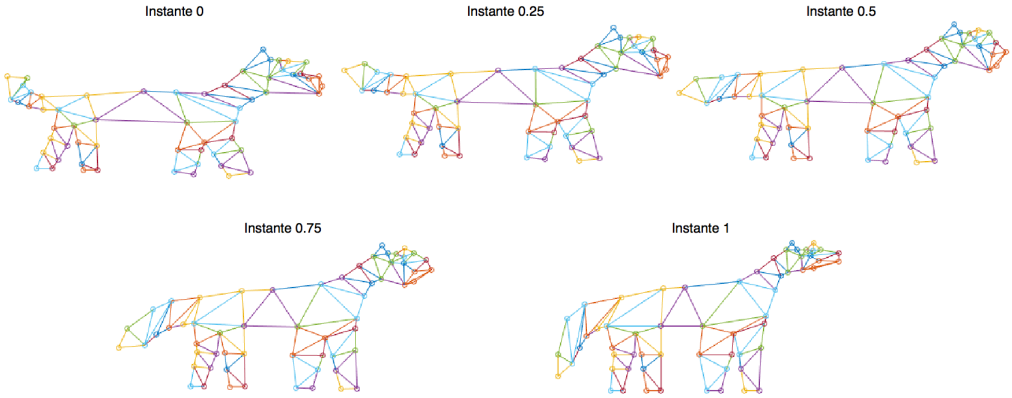
Figura 4.24: Exemplo 7: Melhoria com maximização da área mínima



(a) Interpolação Linear



(b) Interpolação Rígida Média



(c) Interpolação Rígida Ótima

Figura 4.25: Exemplo 7: Morphing

As Figuras 4.22, 4.23, 4.24 e 4.25 apresentam um exemplo para simular a transformação de animais. A Triangulação Compatível foi obtida com recurso a quatro Vértices de Steiner, todos adicionados para triangular a cabeça do animal, zona esta à qual é feito um destaque nas imagens para que seja possível ver os resultados com maior detalhe. A malha inicial contém triângulos finos e longos. Ambas as técnicas de Suavização produzem melhores resultados, no entanto a maximização dos ângulos mínimos produz melhores resultados, uma vez que

gera uma malha de triângulos mais uniforme. Em relação ao *Morphing*, este é um exemplo particularmente interessante, principalmente para verificar como é feita a transformação da cauda do animal por estarem em formas diferentes onde é necessária um movimento em rotação para gerar a transformação correta. A Interpolação Linear volta a falhar, gerando interseções. Ambas as técnicas de Interpolação Rígida produzem bons resultados, onde é de salientar que além de conseguirem novamente evitar interseções, também conseguem simular o movimento natural que esta transformação deve executar.

A Tabela 4.1 mostra os tempos de execução e o número de Vértices de Steiner adicionados em cada um dos exemplos apresentados.

Exemplo	Número de Vértices	Vértices de Steiner Adicionados	Tempo de execução (Segundos)		
			Triangulação Compatível	Suavização	
				Ângulo Mínimo	Área Mínima
1	4	0	0,06	----------	----------
2	6	1	0,32	13,92	11,56
3	6	2	0,44	32,16	44,87
4	20	4	3,76	350,98	529,55
5	15	3	5,20	110,91	93,51
6	12	0	0,55	----------	----------
7	56	4	18,49	270,29	398,56

Tabela 4.1: Detalhes de execução da Triangulação Compatível

A Tabela 4.2 mostra os tempos de execução para os vários tipos de interpolação implementados. Estes tempos são referentes a onze transformações consecutivas, mais concretamente à execução de transformações a cada 0.1 desde o instante 0.0 até ao instante 1.0. Mais abaixo, na Figura 4.26, está um gráfico que compara a média dos tempos que cada uma das técnicas de interpolação obteve na execução de todos os exemplos referidos anteriormente.

Exemplo	Número de Triângulos	Tempo de execução (Segundos)				
		Interpolação				
		Linear	Rígida Média		Rígida Ótima	
			S/ Consistência	C/ Consistência	S/ Consistência	C/ Consistência
1	2	0,94	1,45	1,56	1,03	1,18
2	6	1,13	4,10	4,85	1,33	1,48
3	8	1,31	5,24	5,52	1,62	1,61
4	30	3,07	18,81	19,78	3,53	3,63
5	19	1,79	13,38	13,24	2,61	2,69
6	10	1,26	6,08	6,41	1,52	1,56
7	62	3,93	42,56	45,33	5,84	5,96

Tabela 4.2: Detalhes de execução do *Morphing*

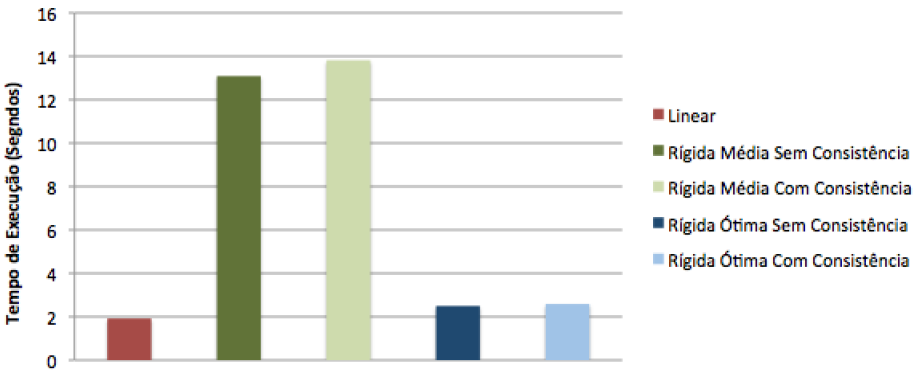


Figura 4.26: Comparação dos tempos de execução

Sobre a Triangulação Compatível, um dos pontos principais é que o algoritmo gera triangulações válidas de forma eficaz, uma vez que tem baixos tempos de execução. No entanto podem existir casos onde a qualidade da malha gerada não é a ideal, devido à formação de triângulos finos e longos. Um exemplo disso é verificado na Figura 4.8, onde o triângulo que une os vértices 2 e 3 com o Vértice de Steiner superior é tão fino que à primeira vista é difícil de notar a sua existência, situação que é resolvida depois da aplicação da Suavização na malha. Sobre a Suavização, é possível comprovar visualmente as melhorias, onde a maximização dos ângulos mínimos é a técnica que gera uma melhores resultados, uma vez que a maximização das áreas mínimas tende a gerar triângulos mais longos e finos, como pode ser verificado no exemplo da Figura 4.14a. Esta situação também pode acontecer com a maximização dos ângulos mínimos, mas é mais esporádica. O algoritmo de Suavização é eficaz, no entanto não está optimizado para ser eficiente, podendo essa situação ser verificada na Tabela 4.1, onde para um baixo número de Vértices de Steiner tem um desempenho razoável, mas agrava-se com o aumento do número de Vértices de Steiner, como expectável. É importante referir que, por vezes, podem existir fatores pontuais que prejudicam a eficiência do algoritmo, como, por exemplo, dificuldades em encontrar um Vértice de Steiner para ligar o par de vértices mais próximos. Esta situação vai fazer com que este par seja ignorado e se tente com o par seguinte da lista, situações que são prejudiciais na eficiência do algoritmo. É possível comprovar esta situação comparando os tempos de execução dos Exemplos 4 e 5, onde o Exemplo 5, apesar deste ter menos vértices, tem um tempo de execução maior que o Exemplo 4 e isso deveu-se a dificuldades em encontrar Vértices de Steiner válidos neste último, o que levou a considerar pares de vértices alternativos.

Em relação ao *Morphing*, os resultados mostram que existem melhorias significativas quando são utilizadas as técnicas de Interpolação Rígida, que produzem melhores transformações que a Interpolação Linear. Foi possível comprovar que a Interpolação Rígida Ótima pode causar deformações quando são utilizadas malhas com triângulos finos e longos, como é possível de verificar nas Figuras 4.19c e 4.21c. Sobre os tempos de execução, expressos sob a forma de gráfico na Figura 4.26, é possível comprovar que existem elevadas diferenças entre os dois tipos

de Interpolação Rígida. Os elevados tempos de execução obtidos pela Interpolação Rígida Média são devido às várias escritas em ficheiro que são efetuadas ao longo da sua execução, que resultam num fator negativo. No entanto é preciso salientar os excelentes tempos de execução obtidos pela Interpolação Rígida Ótima, que conseguem ser muito próximos aos da Interpolação Linear, o que é algo notável, principalmente pelas diferenças de complexidade entre ambas as técnicas. Em relação à ocorrência de interseções, nos exemplos apresentados não existiu nenhuma. Apesar disso não existe garantia que estas não possam ocorrer, sendo possível afirmar, com bases nos testes realizados, que para a grande maioria dos casos são obtidas transformações sem interseções. A última conclusão que é possível verificar é a pouca diferença nos tempos de execução quando é utilizada a Consistência de Rotações, que os aumenta ligeiramente mas nada de significativo.

4.3 EXEMPLOS COM DADOS REAIS

Nesta secção, é testada a viabilidade da solução apresentada com um conjunto de dados reais. Os dados reais representam o movimento de *Icebergs* ao longo do tempo, capturados através de imagens aéreas, onde são extraídos os polígonos que representam o objeto móvel. Na Figura 4.27 pode observar-se duas imagens aéreas capturadas em diferentes instantes: a primeira no dia 19 de Novembro de 2004 e a segunda no dia 2 de Dezembro de 2004.

(a) Captura no dia 19 de Novembro (b) Captura no dia 2 de Dezembro

Figura 4.27: Capturas aéreas em dois dias diferentes

Para os testes são utilizadas três representações, em diferentes instantes de tempo, do mesmo *Iceberg*: I_1, I_2 e I_3. O objetivo é estimar o movimento da transformação entre I_1 e I_3, usando as técnicas de interpolação implementadas, para que possa ser obtida a transformação intermédia correspondente ao instante I_2. Essa transformação intermédia, que simula a forma de I_2, vai ser comparada com a sua verdadeira representação, onde vão ser calculadas as suas similaridades. Um exemplo representativo destes testes é apresentado na Figura 4.28, onde a tracejado estão os polígonos reais e a azul a transformação calculada de I_2, com uma similaridade de 94%. A fórmula que mede a similaridade entre dois polígonos é apresentada

na Equação 4.1, onde R é a representação obtida pelo *Morphing* e I_2 a sua representação verdadeira.

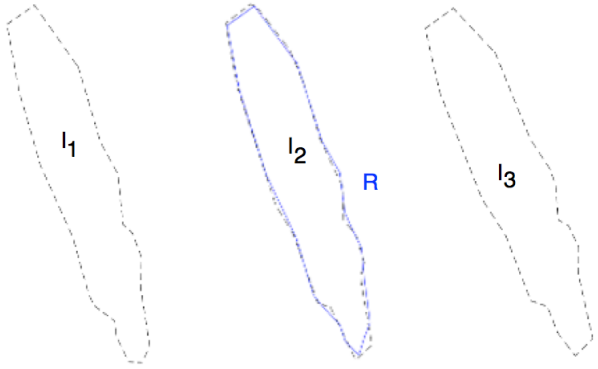
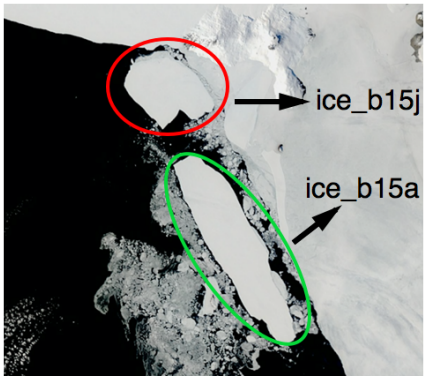


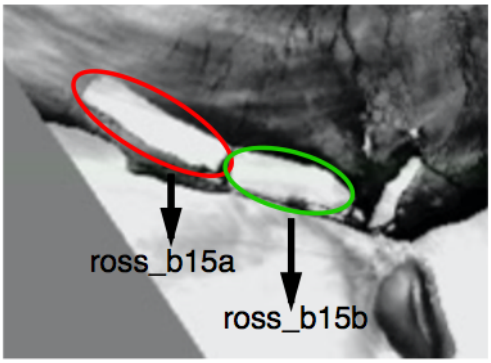
Figura 4.28: Similaridade entre os resultados reais e obtidos

$$Similaridade = \frac{Area(R \cap I_2)}{Area(R \cup I_2)} \tag{4.1}$$

Os *Icebergs* utilizados para os testes foram o *ice_b15a*, *ice_b15j*, *ross_b15a* e *ross_b15b*, que são apresentados na Figura 4.29.



(a) Icebergs ice_b15a e ice_b15j



(b) Icebergs ross_b15a e ross_b15b

Figura 4.29: Icebergs utilizados nos testes

Além da comparação direta dos objetos, as similaridades são testadas também com os seus centroides alinhados. O objetivo deste teste adicional é verificar a similaridade tendo em conta apenas a transformação do objeto, ignorando a sua Translação. Assim é possível avaliar, com maior exatidão, os resultados da transformação e o quanto a Translação influencia os resultados globais.

4.3.1 INTERPOLAÇÃO LINEAR

A Tabela 4.3 apresenta a média dos resultados de similaridade obtidos utilizando a Interpolação Linear, onde a última coluna representa o número de transformações testadas para cada *Iceberg*.

Iceberg	Similaridade (%)		Número de transformações
	Normal	Centrada	
ice_b15a	88,10	92,17	5
ice_b15j	65,74	69,83	8
ross_b15a	77,79	80,77	9
ross_b15b	86,36	89,49	8

Tabela 4.3: Similaridade com Interpolação Linear

Observando os resultados da Tabela 4.3 é possível verificar que a Interpolação Linear obteve melhores resultados com os *Icebergs ice_b15a* e *ross_b15b*. O motivo desses resultados é devido ao facto do movimento desses ser, na sua maioria, translacional. Nos casos onde existe uma transformação com variações significativas ou rotações, como é o caso do *ice_b15j*, os resultados já são piores, como é possível comprovar na Figura 4.30, que apresenta um exemplo de testes de similaridade para este *Iceberg*. Neste exemplo, I_1 corresponde à forma do *Iceberg* no dia 26 de Novembro, I_3 à forma no dia 20 de Dezembro e a transformação intermédia, I_2 ao dia 4 de Dezembro, sendo para esta apresentados os resultados normais e com os centroides alinhados. É possível verificar que o polígono resultante, R, tende a encolher e deformar durante a transformação, o que resulta numa similaridade baixa quando comparada com a sua representação real, que neste caso foi de 40% com os centroides na posição original e 46% com estes alinhados.

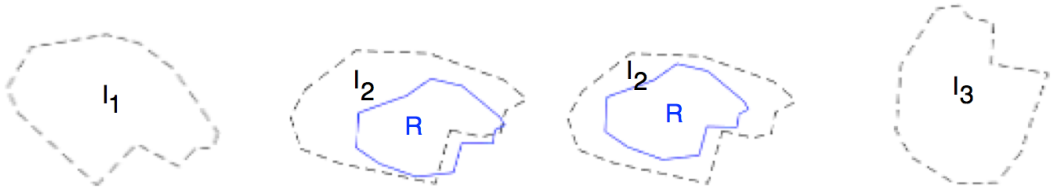


Figura 4.30: Transformação do iceb15_j: Interpolação Linear

4.3.2 INTERPOLAÇÃO RÍGIDA MÉDIA

A Tabela 4.4 apresenta a média dos resultados de similaridade obtidos utilizando a Interpolação Rígida Média.

Iceberg	Similaridade (%)		Número de transformações
	Normal	Centrada	
ice_b15a	87,82	92,16	5
ice_b15j	69,72	77,18	8
ross_b15a	78,78	81,60	9
ross_b15b	86,30	89,46	8

Tabela 4.4: Similaridade com Interpolação Rígida Média

Observando os resultados da Tabela 4.4 é possível comprovar que para os *Icebergs ice_b15a* e *ross_b15b*, que são aqueles cujo movimento ao longo do tempo é na maioria translacional, os resultados foram muito similares aos obtidos com a Interpolação Linear. Os resultados obtidos para o *Iceberg ross_b15a* também foram similares aos anteriores, com uma ligeira melhoria. Os resultados poderiam ter sido melhores, mas foram prejudicados com pequenas deformações que ocorreram durante a transformação. Isso deveu-se à malha de triângulos, uma vez que esta continha alguns triângulos longos e finos, que eram impossíveis de melhorar por não estarem ligados a qualquer Vértice de Steiner. Foi no *Iceberg ice_b15j*, que tem uma elevada rotação ao longo do tempo, onde existiu a melhoria de resultados mais significativa, com um aumento de 7,35% (similaridade centrada) em relação à Interpolação Linear. Na Figura 4.31 são apresentados os testes de similaridade, com a Interpolação Rígida Média, para o mesmo exemplo já utilizado na Figura 4.30. É possível observar as melhorias na transformação, comparando com a Interpolação Linear, uma vez que o polígono resultante, R, encolhe menos e não deforma tanto ao longo das transformações intermédias. Isto resultou numa melhoria da similaridade com a sua representação real, que neste caso foi de 46% com os centroides na posição original e 59% com estes alinhados, que corresponde a um aumento de 6% e 13%, respetivamente, em relação à Interpolação Linear.

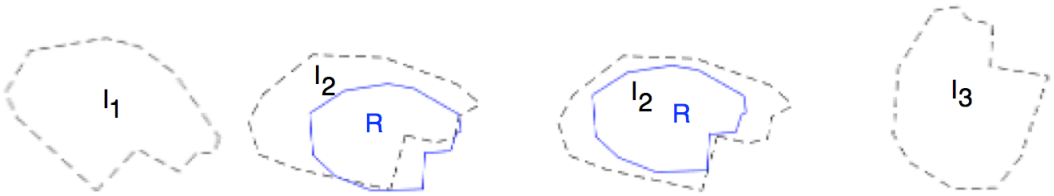


Figura 4.31: Transformação do iceb15_j: Interpolação Rígida Média

4.3.3 INTERPOLAÇÃO RÍGIDA ÓTIMA

A Tabela 4.5 apresenta a média dos resultados de similaridade obtidos utilizando a Interpolação Rígida Ótima.

Iceberg	Similaridade (%)		Número de transformações
	Normal	Centrada	
ice_b15a	85,28	88,56	5
ice_b15j	75,09	90,37	8
ross_b15a	78,90	81,54	9
ross_b15b	85,88	88,94	8

Tabela 4.5: Similaridade com Interpolação Rígida Ótima

Os resultados da Tabela 4.5 mostram que para os *Icebergs* com movimento maioritariamente translacional, *ice_b15a* e *ross_b15b*, as percentagens de similaridade baixaram um pouco. Para o *Iceberg ross_b15a* os resultados foram idênticos ao obtidos anteriormente, onde também foram prejudicados pela qualidade da malha de triângulos. Para o *Iceberg ice_b15j* os resultados foram excelentes, tendo a similaridade centrada aumentado 20,54% e 13,19% quando comparada com a Interpolação Linear e Rígida Média, respetivamente. Na Figura 4.32 é possível comprovar os resultados obtidos por esta técnica para o mesmo exemplo apresentado anteriormente. O polígono resultante, R, não encolheu nem deformou ao longo das transformações intermédias, conseguindo assim uma transformação mais natural e próxima da sua representação real. Isto resultou numa melhoria das similaridades, que foram de 54% com os centroides na posição original e 87% com estes alinhados, o que corresponde a um aumento de 8% e 28%, respetivamente, em relação à Interpolação Rígida Média.

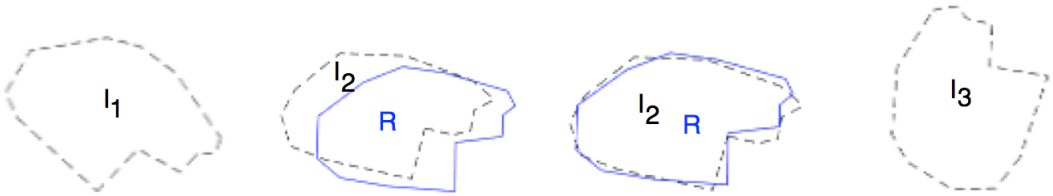


Figura 4.32: Transformação do iceb15_j: Interpolação Rígida Ótima

4.3.4 DISCUSSÃO DOS RESULTADOS

Um dos primeiros pontos a destacar são os bons resultados obtidos pela Interpolação Rígida Ótima na generalidade em todas as transformações calculadas, como é possível verificar na Figura 4.33, onde é apresentado um gráfico com a comparação da média dos resultados de similaridade obtidos pelas várias técnicas de interpolação para todos os *Icebergs* testados.

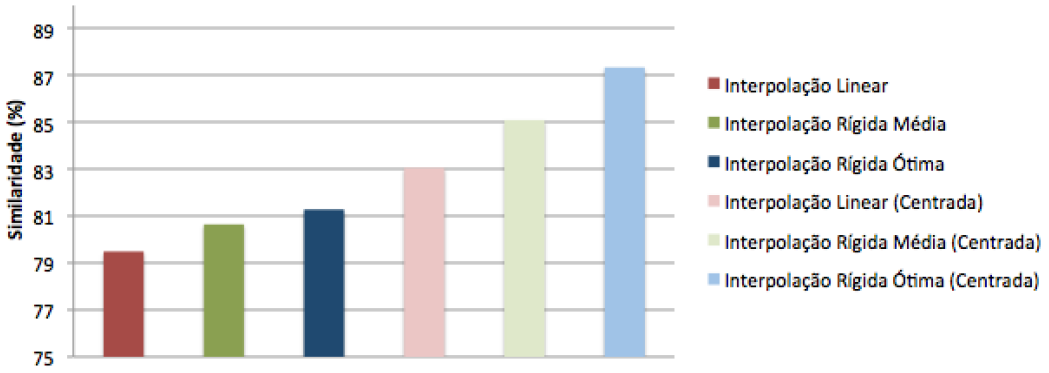


Figura 4.33: Resultados de similaridade

Dentro destes resultados, pode destacar-se os obtidos para o *Iceberg ice_b15j*, que tem uma elevada rotação ao longo do tempo, o que torna mais complexo de estimar o seu movimento. No entanto, através da Interpolação Rígida Ótima, foi possível produzir uma transformação natural ao longo do tempo e muito similar à sua representação real, ao contrário das outras técnicas que geraram resultados com os polígonos a encolherem durante a transformação. Um gráfico de comparação dos resultados para este *Iceberg* é apresentado na Figura 4.34, onde as diferenças de similaridade entre as técnicas de interpolação são evidentes.

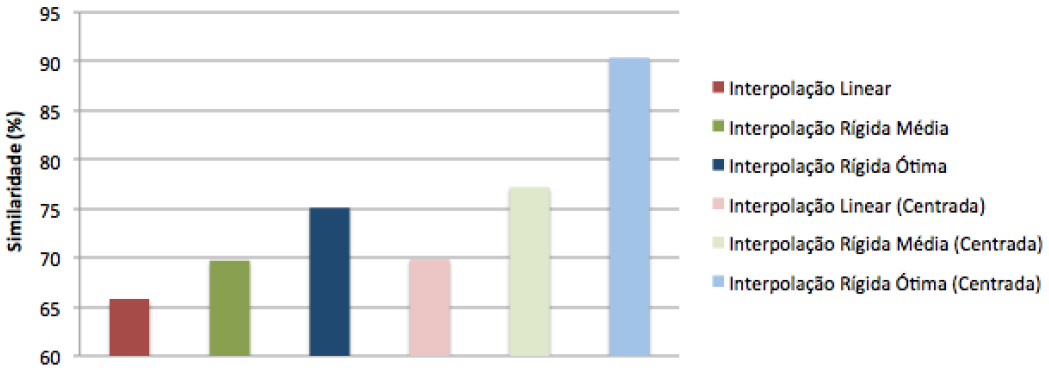


Figura 4.34: Similaridade para Iceberg ice_b15j

Para *Icebergs* com movimentos maioritariamente translacionais, como é o caso do *ice_b15a* e *ross_b15b*, a Interpolação Linear foi a que obteve melhores resultados, o que mostra ser a melhor opção para simular este tipo de movimento. Na Figura 4.35 é apresentado um gráfico de comparação dos resultados para movimentos maioritariamente translacionais.

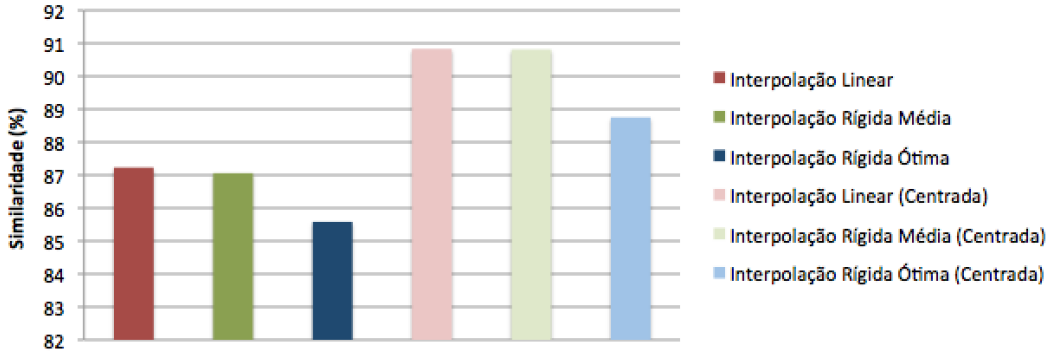


Figura 4.35: Similaridade para movimentos translacionais

Outro ponto importante de referir são as diferenças entre os resultados normais e os resultados com os centroides alinhados. Esta diferença, representada no gráfico da Figura 4.36, está relacionada com a Interpolação da Translação, que é feita linearmente ao longo do movimento, enquanto que o movimento dos *Icebergs* não é linear.

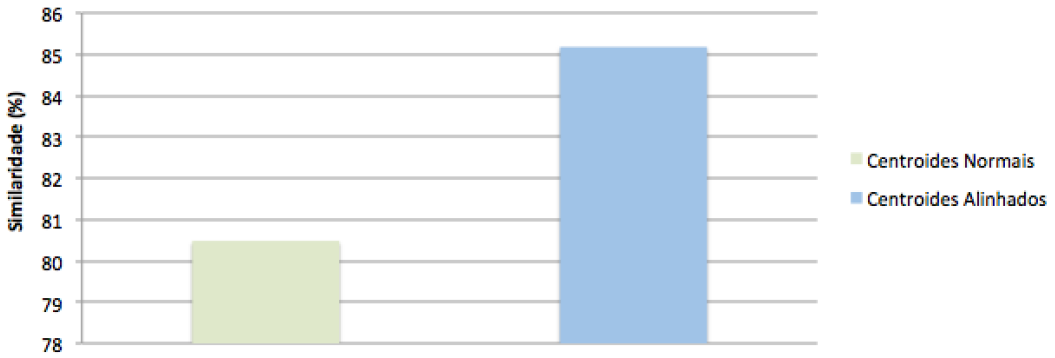


Figura 4.36: Diferenças entre centroides normais e alinhados

Por último, é importante destacar a necessidade de uma boa malha de triângulos para obter bons resultados com as técnicas de Interpolação Rígida. Um exemplo desta situação foram os resultados obtidos para o *Iceberg ross_b15a*, expressos em gráfico na Figura 4.37. Apesar da Interpolação Rígida continuar a obter melhores resultados, estes poderiam ser melhores se tivesse sido utilizada uma malha com boa qualidade.

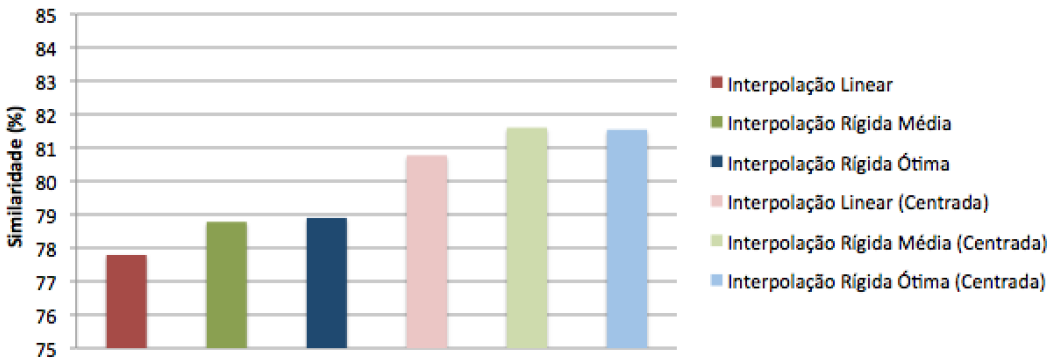


Figura 4.37: Similaridade para Iceberg ross_b15a

capítulo 5

Considerações Finais

5.1 conclusões e contributos

Esta Dissertação consistiu na realização de um trabalho exploratório sobre a representação do comportamento espácio-temporal de objetos móveis em bases de dados e sistemas de informação geográfica. O principal objetivo foi estudar a viabilidade de usar técnicas de *Morphing* baseadas em Triangulação Compatível para representar a transformação de objetos móveis ao longo do tempo. Este objetivo foi concretizado através da implementação de um conjunto de algoritmos, o desenvolvimento de programa de teste, denominado T-Morph, e a realização diversos testes para avaliar e comparar a qualidade das soluções desenvolvidas.

Na componente de Triangulação Compatível, foi utilizado o método proposto por *Surazhsky et al* [15] que é baseada em partições de polígonos através de ligações entre os pares de vértices mais próximos, introduzindo Vértices de Steiner apenas quando necessário. Os resultados obtidos por esta técnica mostram que a qualidade da malha de triângulos por vezes pode não ser a melhor, sendo necessário fazer uma Suavização da malha para obter os melhores resultados. Para isso foi desenvolvido um algoritmo de Suavização, onde é possível escolher o fator de maximização: ângulos mínimos ou áreas mínimas. O problema desta componente é que podem existir situações onde, mesmo depois da Suavização, a malha continua com fraca qualidade. Os melhores resultados são obtidos quando é efectuada uma Triangulação Compatível, seguido de uma Suavização com maximização dos ângulos mínimos.

Em relação ao *Morphing*, o principal objetivo é conseguir representar uma transformação o mais natural possível e sem interseções, sendo para isso utilizados os métodos propostos por *Alexa et al* [6] e *Baxter et al* [25]. Ests métodos utilizam uma Interpolação Rígida com o

objetivo de preservar as propriedades físicas dos objetos, produzindo assim uma transformação mais natural. No entanto esta técnica gera posições diferentes para os vértices partilhados por triângulos ao longo da transformação, sendo que aqui foram implementadas duas soluções para a obtenção da posição final: a primeira calcula a posição final dos vértices tendo em conta a média das várias coordenadas (Interpolação Rígida Média) e a segunda segue a proposta de *Baxter et al* [25], que calcula a posição final com recurso a equações normais (Interpolação Rígida Ótima).

Globalmente, os resultados são promissores e mostram que a solução explorada neste trabalho permite dar resposta a algumas das lacunas identificadas em trabalhos anteriores. Em particular, os resultados mostram que a técnica de Interpolação Rígida, desenvolvida neste trabalho, permite obter representações mais realistas do que as técnicas propostas na literatura em bases de dados espácio-temporais. Além disso, este trabalho mostra que é possível evitar problemas topológicos na representação da transformação dos objetos ao longo do tempo, por exemplo, a interseção de arestas, usando, por exemplo, técnicas de suavização nas malhas de triângulos. No entanto, não é possível garantir que a topologia das geometrias que representam a evolução dos objetos seja sempre válida. Estas situações podem acontecer quando as geometrias que definem a forma dos objetos no momento inicial e no momento final de uma transformação são muito complexas e díspares.

5.2 TRABALHO FUTURO

Os resultados mostram que é possível melhorar. Foi possível comprovar a influência da malha de triângulos nos resultados, onde uma malha com fraca qualidade resulta em transformações deformadas. Neste aspeto, o algoritmo utilizado na Triangulação Compatível (*Surazhsky et al* [15]) está limitado, pois apenas adiciona Vértices de Steiner quando necessário e isso pode gerar malhas de fraca qualidade que a Suavização não consegue melhorar. É necessário utilizar um algoritmo que consiga construir malhas de triângulos uniformes e de boa qualidade, mesmo que isso implique a adição de mais Vértices de Steiner e, por consequência, o aumento da complexidade. Foram apresentadas alternativas na Secção 2.2.3.

Outra limitação já identificada é o algoritmo para a Suavização da malha de triângulos. Este foi feito com uma abordagem básica apenas com o objetivo de ser eficaz, não sendo as questões de eficiência uma prioridade durante a sua realização. Existem vários estudos, já referidos na Secção 2.2, que irão conseguir melhorar os resultados atuais.

A solução atual não dá garantia total em relação às interseções, no entanto consegue evitar a maioria dos casos onde estas existem. A garantia total também é um ponto que deve ser estudado no futuro.

Referências

[1] R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider e M. Vazirgiannis, «A foundation for representing and querying moving objects», *ACM Trans. Database Syst.*, vol. 25, nº 1, pp. 1–42, mar. de 2000, ISSN: 0362-5915. DOI: 10.1145/352958.352963. endereço: http://doi.acm.org/10.1145/352958.352963.

[2] N. Pelekis, Y. Theodoridis, S. Vosinakis e T. Panayiotopoulos, «Hermes – a framework for location-based data management», em *Proceedings of the 10th International Conference on Advances in Database Technology*, sér. EDBT'06, Munich, Germany: Springer-Verlag, 2006, pp. 1130–1134, ISBN: 3-540-32960-9, 978-3-540-32960-2. DOI: 10.1007/11687238_75. endereço: http://dx.doi.org/10.1007/11687238_75.

[3] L. Zhao, P. Jin, L. Zhang, H. Wang e S. Lin, «Developing an oracle-based spatio-temporal information management system», English, em *Database Systems for Adanced Applications*, sér. Lecture Notes in Computer Science, J. Xu, G. Yu, S. Zhou e R. Unland, eds., vol. 6637, Springer Berlin Heidelberg, 2011, pp. 168–176, ISBN: 978-3-642-20243-8. DOI: 10.1007/978-3-642-20244-5_16. endereço: http://dx.doi.org/10.1007/978-3-642-20244-5_16.

[4] L. Matos, J. Moreira e A. Carvalho, «A spatiotemporal extension for dealing with moving objects with extent in oracle 11g», *SIGAPP Appl. Comput. Rev.*, vol. 12, nº 2, pp. 7–17, jun. de 2012, ISSN: 1559-6915. DOI: 10.1145/2340416.2340417. endereço: http://doi.acm.org/10.1145/2340416.2340417.

[5] L. Liu, G. Wang, B. Zhang, B. Guo e H.-Y. Shum, «Perceptually based approach for planar shape morphing», em *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*, out. de 2004, pp. 111–120. DOI: 10.1109/PCCGA.2004.1348341.

[6] M. Alexa, D. Cohen-Or e D. Levin, «As-rigid-as-possible shape interpolation», *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pp. 157–164, 2000, ISSN: 10473203. DOI: 10.1145/344779.344859. endereço: http://portal.acm.org/citation.cfm?doid=344779.344859.

[7] E. Tøssebro e R. H. Güting, «Creating representations for continuously moving regions from observations», em *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, sér. SSTD '01, London, UK, UK: Springer-Verlag, 2001, pp. 321–344, ISBN: 3-540-42301-X. endereço: http://dl.acm.org/citation.cfm?id=647227.719098.

[8] M. Mckennney e R. Frye, «Generating moving regions from snapshots of complex regions», *ACM Trans. Spatial Algorithms Syst.*, vol. 1, nº 1, 4:1–4:30, jul. de 2015, ISSN: 2374-0353. DOI: 10.1145/2774220. endereço: http://doi.acm.org/10.1145/2774220.

[9] L. Paulo, «Aplicação de técnicas de morphing em bases de dados espacio-temporais», tese de mestrado, Universidade de Aveiro, 2012.

[10] P. Mesquita, «Técnicas de morphing para representação de objetos móveis geográficos», tese de mestrado, Universidade de Aveiro, 2013.

[11] *Simplex - wikipedia, the free encyclopedia.* endereço: http://en.wikipedia.org/wiki/Simplex (acedido em 01/09/2015).

[12] G. A. Hansen, R. W. Douglass e A. Zardecki, *Mesh enhancement.* Imperial College Press, 2005.

[13] H. Xu e T. S. Newman, «An angle-based optimization approach for 2d finite element mesh smoothing», *Finite Elem. Anal. Des.*, vol. 42, nº 13, pp. 1150–1164, 2006, ISSN: 0168-874X. DOI: 10.1016/j.finel.2006.01.016. endereço: http://dx.doi.org/10.1016/j.finel.2006.01.016.

[14] T. Zhou e K. Shimada, «An angle-based approach to two-dimensional mesh smoothing», *Proceedings of the 9th International Meshing Roundtable*, nº 412, pp. 373–384, 2000. endereço: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.9683%5C&rep=rep1%5C&type=pdf.

[15] V. Surazhsky e C. Gotsman, «High quality compatible triangulations», *Engineering with Computers*, vol. 20, nº 2, pp. 147–156, 2004, ISSN: 01770667. DOI: 10.1007/s00366-004-0282-6.

[16] M. Ebene Ebene, Y. Maréchal, D. Armand e D. Ladas, «An adaptive remeshing technique insuring high quality meshes», *IEEE Transactions on Magnetics*, vol. 44, pp. 1222–1226, 2008. endereço: https://hal.archives-ouvertes.fr/hal-00382730.

[17] *Dynamic 2d navigation mesh generation using medial axis and voronoi regions | the orange day.* endereço: http://www.theorangeday.com/experiments/dynamic-2d-navigation-mesh-generation-using-medial-axis-and-voronoi-regions/ (acedido em 10/09/2015).

[18] F. Razafindrazaka, «Delaunay triangulation algorithm and application to terrain generation», pp. 2–12, 2009.

[19] *Delaunay triangulation - wikipedia, the free encyclopedia.* endereço: https://en.wikipedia.org/wiki/Delaunay%5C_triangulation (acedido em 10/09/2015).

[20] D. T. Lee e B. J. Schachter, «Two algorithms for constructing a delaunay triangulation», *International Journal of Computer & Information Sciences*, vol. 9, nº 3, pp. 219–242, 1980, ISSN: 00917036. DOI: 10.1007/BF00977785.

[21] B. Aronov, R. Seidel e D. Souvaine, «On compatible triangulations of simple polygons», *Comput. Geom. Theory Appl.*, vol. 3, nº 1, pp. 27–35, 1993, ISSN: 0925-7721. DOI: 10.1016/0925-7721(93)90028-5. endereço: http://dx.doi.org/10.1016/0925-7721(93)90028-5.

[22] E. Kranakis e J. Urrutia, «Isomorphic triangulations with small number of steiner points», *International Journal of Computational Geometry & Applications*, vol. 9, nº 2, pp. 171–180, 1999.

[23] H. Gupta e R. Wenger, «Constructing piecewise linear homeomorphisms of simple polygons», *Journal of Algorithms*, vol. 22, nº 1, pp. 142–157, 1997, ISSN: 01966774. endereço: http://www.sciencedirect.com/science/article/pii/S0196677485708085.

[24] M. McKenney e J. Webb, «Extracting moving regions from spatial data», em *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, sér. GIS '10, San Jose, California: ACM, 2010, pp. 438–441, ISBN: 978-1-4503-0428-3. DOI: 10.1145/1869790.1869856. endereço: http://doi.acm.org/10.1145/1869790.1869856.

[25] W. Baxter, P. Barla e K. Anjyo, «Rigid shape interpolation using normal equations», *NPAR '08 Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, pp. 59–64, 2008. DOI: http://doi.acm.org/10.1145/1377980.1377993. endereço: http://dl.acm.org/citation.cfm?id=1377993.

[26] M. Shapira e a. Rappoport, «Shape blending using the star-skeleton representation», *IEEE Computer Graphics and Applications*, vol. 15, nº 2, pp. 44–50, 1995, ISSN: 02721716. DOI:

10.1109/38.365005. endereço: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=365005.

[27] T. W. Sederberg, P. Gao, G. Wang e H. Mu, «2dd shape blending: An intrinsic solution to the vertex path problem», em *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, sér. SIGGRAPH '93, Anaheim, CA: ACM, 1993, pp. 15–18, ISBN: 0-89791-601-8. DOI: 10.1145/166117.166118. endereço: http://doi.acm.org/10.1145/166117.166118.

[28] D. Xu, H. Zhang, Q. Wang e H. Bao, «Poisson shape interpolation», em *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, sér. SPM '05, New York, NY, USA: ACM, 2005, pp. 267–274, ISBN: 1-59593-015-9. DOI: 10.1145/1060244.1060274. endereço: http://doi.acm.org/10.1145/1060244.1060274.

[29] C. Gotsman e V. Surazhsky, «Guaranteed intersection-free polygon morphing», *Computers and Graphics (Pergamon)*, vol. 25, nº 1, pp. 67–75, 2001, ISSN: 00978493. DOI: 10.1016/S0097-8493(00)00108-4.

[30] K. Shoemake, M. Hill e T. Duff, «Matrix animation and polar decomposition», *Matrix*, vol. 92, pp. 258–264, 1992, ISSN: 07135424. DOI: 10.1.1.56.1336. endereço: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.1336%5C&rep=rep1%5C&type=pdf.

[31] G. H. Golub e C. F. Van Loan, *Matrix Computations (3rd Ed.)* Baltimore, MD, USA: Johns Hopkins University Press, 1996, ISBN: 0-8018-5414-8.

[32] *Singular value decomposition - matlab svd*. endereço: http://www.mathworks.com/help/matlab/ref/svd.html (acedido em 17/09/2015).

[33] R. Penrose e J. A. Todd, «On best approximate solutions of linear matrix equations», *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 52, nº 01, pp. 17–19, out. de 2008, ISSN: 0305-0041. DOI: 10.1017/S0305004100030929. endereço: http://www.journals.cambridge.org/abstract%5C_S0305004100030929.

[34] *Fit geometric transformation to control point pairs - matlab fitgeotrans*. endereço: http://www.mathworks.com/help/images/ref/fitgeotrans.html (acedido em 25/09/2015).

[35] *Four-quadrant inverse tangent - matlab atan2*. endereço: http://www.mathworks.com/help/matlab/ref/atan2.html (acedido em 03/10/2015).

[36] *Correct phase angles to produce smoother phase plots - matlab unwrap*. endereço: http://www.mathworks.com/help/matlab/ref/unwrap.html (acedido em 07/10/2015).